



Budapest University of Technology and Economics  
Faculty of Electrical Engineering and Informatics  
Department of Telecommunications and Telematics

# Design and Implementation of Distributed Applications in Ad Hoc Network Environment

**Master's Thesis**

Balázs Kovács

Advisors:

Miklós Aurél Rónai

*M.Sc., Ericsson Research, Traffic Lab*

Zoltán Richárd Turányi

*M.Sc., Ericsson Research, Traffic Lab*

Róbert Szabó

*Ph.D., Budapest University of Technology and Economics*

Budapest, 2003.

# Nyilatkozat

Alulírott Kovács Balázs, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Budapest, 2003. május

.....  
Kovács Balázs

## Kivonat

Manapság az ad hoc hálózatok népszerű kutatási területnek számítanak. Léteznek olyan technológiák, melyek lehetővé teszik mobil eszközök számára a központi felügyelet nélküli ad hoc hálózatok kialakítását. Folyamatosan tágul azoknak az eszközöknek a köre, melyek képesek ilyen jellegű hálózat kialakítására. Mivel a digitális személy asszisztensek (PDA) és mobiltelefonok képességei egyre inkább fejlődnek, eme eszközök is alkalmasak arra, hogy részt vegyenek egy ad hoc hálózat kialakításában. Nemcsak kis méretük vonzó, segítségükkel az emberek bárhol konzisztensen hozzáférhetnek személyes adataikhoz. Ugyanakkor a hordozható eszközöknek egyik nagy hátránya éppen a kis méretükből adódik.

Felmerült az igény, hogy hagyományos méretű perifériákat csatlakoztathassunk a hordozható eszközökhöz, például egy kényelmes billentyűzetet, egeret, vagy egy élvezhető méretű megjelenítőt, valamint hogy kiegészítsük képességeiket külső szolgáltatások segítségével, például egy nyomtatóval. A hordozható eszközöket különböző célokra tervezték és ezért különböző képességbeli korlátokkal rendelkeznek. A dinamikus architektúrájú hálózatok és az ezekben résztvevő változatos eszközök nehézkessé teszik az ad hoc hálózatokra történő alkalmazás-fejlesztést, mivel a hálózati erőforrások és szolgáltatások előre nem ismertek.

Két lehetőség merül fel ennek a problémának a megoldására. Az egyik, hogy az eszközök képességeihez kell formálni alkalmazásainkat és így sok különböző változatot kell készíteni. A másik lehetőség egy middleware kifejlesztése, mely képes az eszközök hiányzó képességeit kiegészíteni más eszközök szolgáltatásaival. Így az alkalmazás-fejlesztők számára lehetőség nyílna, hogy statikusan jelöljék meg alkalmazásaik erőforrásigényeit a middleware felett, és ezen igények később dinamikusan összerendelhetőek lennének a szolgáltatásokkal.

A Blown-up middleware technológia eme probléma megoldására született. Nemcsak leegyszerűsíti az ad hoc hálózatokra történő alkalmazás-fejlesztést, hanem lehetővé teszi a felhasználók számára, hogy igénybevegyék a Blown-up alkalmazásokat ún. kapcsolatrendszerekben, és ezzel kialakítsák saját személyi hálózataikat.

Diplomamunkámban bemutatom a számítástechnika egy területét, a *pervasive* vagy *ubiquitous computing*-ot, és megmutatom, hogy a Blown-up elosztott szolgáltatás elérési technológiát hogyan lehet beilleszteni eme elképzelésekbe, mik azok a problémák melyek a Blown-uppal megoldhatóvá válnak. Bemutatom a middleware-hez általam megvalósított első alkalmazásokat és a segítségükkel kialakítható kapcsolatrendszert. Szintén bemutatok egy általam készített vezérlő alkalmazást és annak grafikus felhasználói felületét mely lehetővé teszi a Blown-up kapcsolatrendszerek egyszerű szervezését. A megvalósított alkalmazások mindegyike a Blown-up Micronet Protocol programozói felületét használja. A diplomamunkában kiemelem a Blown-up előnyeit és azon jellemzőit, melyek további fejlesztést igényelnek ahhoz, hogy a middleware teljesebb és még használhatóbb legyen.

## Abstract

Nowadays, ad hoc networks are an emerging field of research. There are existing network technologies that enable mobile devices to form an ad hoc network without centralized administration. The set of devices that are capable of being involved in such a network is expanding. However, most of these devices are likely to be small handhelds, like Personal Digital Assistants (PDA) or cellular phones, as their capabilities are continuously making progress. Compact size is not the only attractive characteristic of handhelds, with the help of handhelds, people can also keep their personal data nearby, and access them consistently any time. The only handicap of handheld devices arises from their small size, namely that they have small data input (e.g. keyboard or mouse) and output (display) interface.

To avoid the problems of small size, a claim emerged to attach normally-sized peripheral devices to handhelds like a comfortable keyboard, mouse, a display of enjoyable size, or to complete their capabilities by connecting them to standalone services, for example to a printer. Handheld devices may be designed for different special operations with different capability restrictions. Overall, we can say that application development over such a highly dynamic network or device architecture is complicated, because available network services are not foreseeable.

Two opportunities arise to solve this problem. The first one is to make our applications adapt to device capabilities, thus, creating more versions. The other one is to develop some kind of a middleware that is able to assign missing device capabilities to other devices that have the requested capability. Hence, application developers could be able to register static resource requests over this middleware, and later these requests could be dynamically connected to the service provider devices in the ad hoc network.

*Blown-up* technology has been developed by following the latter guideline. This middleware not only aims simplifying application development over the scenario described above, but also makes the Blown-up enabled applications available for users in set of sessions, thus allowing people to create their own Personal Area Networks (PAN) over ad hoc networks.

In my thesis I introduce a field of computer technology, called *pervasive* or *ubiquitous computing* and trace how the Blown-up distributed service access technology can be fit into these visions, and what the problems are that can be solved by Blown-up. I present the first applications I implemented for this middleware and demonstrate a session with the help of these applications. I also present a control application and its graphical user interface through which, users are able to manage the sessions of Blown-up easily. Each of the implemented applications use the Blown-up Micronet Protocol application programming interface (API). I illustrate the advantages of Blown-up and those features that may need correction or further development, in order to make this middleware more useful.

# Acknowledgements

I would like to thank all my colleagues for their support, especially for Gergely Biczók, Kristóf Fodor and Ágoston Szabó.

Special thanks to Miklós Rónai (Hawai), who have supported me and my colleagues in the last two years sparing no pains and time. I would also specially thank to András Valkó and Zoltán Turányi for their precious help.

I would like to thank my family for supporting me especially in the last five years.

I would like to thank Móni who stood by me all the time.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Mark Weiser's Vision . . . . .	2
1.2	Expectations and Challenges . . . . .	4
1.3	A Vision Scenario . . . . .	6
1.4	Structure of the Thesis . . . . .	8
<b>2</b>	<b>Related Work</b>	<b>10</b>
2.1	Oxygen . . . . .	10
2.2	Portolano . . . . .	14
2.3	System Support for Pervasive Environments . . . . .	19
2.4	Problem Statement . . . . .	22
<b>3</b>	<b>The Blown-up Middleware</b>	<b>24</b>
3.1	Goals and Assumptions . . . . .	25
3.2	System Architecture and Protocol Stack . . . . .	28
3.2.1	Overview of the Protocol Stack . . . . .	30
3.2.2	User Plane . . . . .	32
3.2.3	Control Plane . . . . .	34
3.3	Application Programming Interface . . . . .	36
<b>4</b>	<b>Applications for BUMP</b>	<b>40</b>

4.1	Blown-up User Applications . . . . .	40
4.1.1	File Access Service . . . . .	42
4.1.2	MP3 Service . . . . .	45
4.1.3	Summary of BUMP User Plane . . . . .	47
4.2	Control Application . . . . .	48
4.2.1	Back-end . . . . .	50
4.2.2	Front-end . . . . .	55
4.2.3	Summary of BUMP Control Plane . . . . .	58
<b>5</b>	<b>Analysis of BUMP</b>	<b>60</b>
<b>6</b>	<b>Conclusion</b>	<b>63</b>
<b>A</b>	<b>API Functions and Structures</b>	<b>65</b>
A.1	Description of API Functions . . . . .	65
A.2	Description of API Structures . . . . .	68

# List of Abbreviations

ACK – Acknowledgement

AFS – Andrew File System

AODV – Ad hoc On-demand Distance Vector

API – Application Programming Interface

BUMP – Blown-Up Micronet Protocol

BUMPC – BUMP controller

CA – Control Application

CAC – Context Aware Computing

CORBA – Common Object Request Broker Architecture

CSMACA – Carrier Sense Multiple Access with Collision Avoidance

DFS – Distributed File System

DSDV – Destination-Sequenced Distance Vector

DSR – Dynamic Source Routing

E21 – Enviro21

FTP – File Transfer Protocol

GUI – Graphical User Interface

H21 – Handy21

HAVi – Home Audio Video Interoperability

IC – Integrated Circuit

IDL – Interface Description Language

IEEE – Institute of Electrical and Electronics Engineers

IP – Internet Protocol

MAC – Medium Access Control NFS – Network File System



N21 – Network21  
PAN – Personal Area Network  
PDA – Personal Digital Assistant  
QoS – Quality of Service  
RDP – Resource Discovery Protocol  
RF – Radio Frequency  
RMI – Remote Method Invocation  
RPC – Remote Procedure Call  
RSVP – Resource Reservation Protocol  
SDL – Specification Description Language  
SLP – Service Location Protocol  
TAP – Transport Access Point  
TCL/TK – Tool Command Language/ToolKit  
TCP – Transmission Control Protocol  
TORA – Temporally-Ordered Routing Algorithm  
TP – Transport  
UDP – User Datagram Protocol  
UI – User Interface  
WLAN – Wireless Local Area Network  
XML – Extensible Markup Language

# List of Figures

1.1	Fred's presentation . . . . .	7
2.1	Oxygen technologies . . . . .	11
2.2	System support principles for pervasive applications [GRIMM et al.01] . . . . .	21
3.1	Working easy . . . . .	26
3.2	Time to play! . . . . .	26
3.3	Play against each other . . . . .	27
3.4	System architecture [TDK2002] . . . . .	28
3.5	World seen by applications [TDK2002] . . . . .	29
3.6	BUMP layer structure [TDK2002] . . . . .	31
4.1	Implemented session . . . . .	41
4.2	User interface of the file service client . . . . .	44
4.3	User interface of the mp3 service client . . . . .	46
4.4	UML diagram . . . . .	51
4.5	Class AppPIN . . . . .	52

4.6	Class <code>ActiveSession</code> . . . . .	54
4.7	Device, application and pin lookup on GUI . . . . .	56
4.8	Creating a session, step 1 . . . . .	57
4.9	Creating a session, step 2 . . . . .	58
4.10	Session lookup and delete . . . . .	59

# Chapter 1

## Introduction

Nowadays cellular phones, Personal Digital Assistants (PDA) are increasingly spreading. The capabilities of these devices are continuously making progress, slowly approaching the capabilities of a real personal computer. On the other hand almost all of them fit into a coat pocket, hence we like to use them. They not only take advantage due to their small sizes, but we can also keep our files or works near to ourselves, and regardless of the place of the usage they stay consistent. Unfortunately the major disadvantage of handhelds comes from their major advantage: the small size. Feeding data into these devices generally happens through a tiny keyboard or some touchpad, while graphical data output also appears on a small-sized display.

A claim emerges to connect ordinary large-sized data input, or displaying devices like a simple PC keyboard or PC display to these small devices such as PDAs, cellular phones. Today there are technologies available solving the communication between small devices and their peripheral devices, such a technology is Bluetooth. If we re-think the problem not only simple peripheral devices can be connected to mobile devices. Drafting up universally, services are needed to be connected to other services. Services that run over some kind of hardware (e.g., keyboard, mouse, display) and are able to communicate with the other service using it on a different hardware (e.g., chess game on a PDA).

The mobile devices mentioned above form an ad hoc network, which has a frequently changing architecture: resources and devices come and go. However, application developers

need static resources onto which they can build their programs. In case of a single computer or a statically linked network, developers can create applications that use the resources available on that architecture. In an ad hoc network they can not make the assumptions on certain capabilities, but they can require the presence of them. That is why services can be also connected to other services. Let us assume that we get into a small-ranged Personal Area Network (PAN) consisting of devices with limited capabilities. It can happen that we can use a service only if another service is also available in the network. For example we want to play with a chess game running on a very simple hardware. To start playing, it needs a keyboard service for data input and a display service for graphical output. So before we can use the chess service, we have to find services in the network that meet the requirements of the chess game. If we manage to create this connection system, a new session can be created which can become part of our personal network.

*Blown up concept* [TDK2002] was developed to hide the dynamic architecture of an ad hoc network from application developers. Blown up concept uses the Blown-up Micronet Protocol (BUMP) [Fodor2003] to handle connection systems of services. In my thesis I am going to discuss how to create applications which are able to present the abilities of the Blown-up concept by means of the Application Programming Interface (API) of BUMP. Before I discuss these applications I summarize the background field of Blown-up concept, the *ubiquitous* or *pervasive computing* and trace how Blown-up concept can be fit into these quite abstract visions.

## 1.1 Mark Weiser's Vision

The principles of *ubiquitous computing* were first drafted by Mark Weiser. He described his vision about the computer of the 21st century in an article [Weiser91] published in 1991. He states that information technology should become a natural part of people's everyday lives, usage of devices should be just as evident as, for example, reading. When you read a sign on the street you absorb its information without consciously performing the act of reading. In his opinion "the most profound technologies are those that disappear. They weave themselves into the fabric of our everyday life until they are indistinguishable from it." His conception is the opposite of the paradigm of virtual reality, since the latter

focuses an enormous apparatus on simulating the world rather than on invisibly enhancing the world that already exists. In his opinion *ubiquitous computing* should explore quite different ground from the idea that computers should be autonomous agents that take on our goals [Weiser93]. To characterize the difference he describes an example. Suppose you want to lift a heavy object. You can call in your strong assistant to lift it for you, or you can have yourself made effortlessly, unconsciously, stronger and just lift it. There are times when both are good. Much of the past and current effort for better computers has been aimed at the former; *ubiquitous computing* aims at the latter. The point is: by the help of *ubiquitous computing* we could *focus on tasks but not on the tool*. There has been a big need for developments in the field of mobile computing to actually realize Mark Weiser's conception. *Ubiquitous computing* needs:

- cheap and low power consumption hardware
- some kind of network to connect the devices
- software elements that support distributed operation

At the time he wrote his article the technology needed to implement a system based on his ideas did not yet exist. In the recent decade information technology has come a long way. Cheaper and more powerful hardware, intelligent software elements, faster and mobile networks, and a great number of scientific achievements makes possible to materialize Weiser's idea. These include the use of fiber optics in data transmission [Gilder93], that provides almost limitless bandwidth, the evolution of human voice controlled systems [Tatai97, OGN92] (which is needed for new generation user interfaces) and the breakthrough in image processing [RC93]. Nowadays research based on wireless ad hoc networks are of great interest. Through these researches new technologies are discovered that can be used in the realization of *ubiquitous computing*. Various radio interface technologies were developed for supporting communication: IEEE 802.11 [WLAN99], HiperLAN and HiperLAN2 [HLAN2] and Bluetooth [JH98, BBSpec]. 802.11 uses CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) [Karn90] in ad hoc mode. There are also various routing algorithms like AODV (Ad hoc On-demand Distance Vector), DSR (Dynamic Source Routing) [DPR00], DSDV (Destination Sequenced Distance Vector) [PB94] and TORA (Temporarily Ordered Routing Algorithm) [PC97, BMJHJ98]

In his articles Mark Weiser named this new field of computer science *ubiquitous computing*. However, recent documents refer to the same subject as *pervasive computing*, which naming comes from researchers of IBM. The connection between these two expressions is often mentioned and there are also articles dealing with the matter [McCroory00]. Some consider the two expressions as synonyms, *pervasive computing* is simply a new name of *ubiquitous computing*. Others think that the two expressions mean quite the same with some differences: *pervasive computing* involves devices like handhelds - small, easy-to-use devices - through which people will be able to get information on anything and everything (e.g. surfing on the Internet using a cellular phone), while the goal of *ubiquitous computing* is to hide computers everywhere into the background. In recent documents there is also another name for this field of computer science - *invisible computing*. In this document I consider these expressions as synonyms of each other.

## 1.2 Expectations and Challenges

I examine the challenges of *pervasive computing* based on M. Satyarayanan's article [Satya01] and discuss what relations *pervasive computing* has with distributed systems and mobile computing.

The field of distributed system arose at the intersection of personal computers and local area networks. The research created a conceptual framework and algorithmic base that has proven to be a value in all work involving two or more computers connected by a network. This body of knowledge spans many areas that are foundational to *pervasive computing*. Technologies based on distributed systems support remote and safe communication, fault tolerance, high availability and remote information access.

The appearance of laptop computers and wireless local area networks in the early 90's led researchers to confront the problems that arise in distributed systems with mobile clients. They made new, and modified existing network technologies (Mobile IP [BPT96], ad hoc protocols [BMJHJ98], TCP for wireless networks [BSAK95]), labored mobile information access algorithms, adaptive applications (proxies), system-level energy saving techniques and location aware system behavior to solve these problems.

The main goal of *pervasive computing* is to create a technology that can invisibly assimilate into our everyday lives. Since motion is an integral part of our lives *pervasive computing* must support mobility, otherwise a user will be aware of the technology when he moves. Hence the research agenda of *pervasive computing* incorporates four additional research thrusts.

The first is the usage and integration of smart spaces. Smart spaces are intelligent computer systems installed in common buildings, rooms, etc. When used efficiently, smart spaces are e.g. able to control the buildings features like heating and lighting of rooms according to the people's position and actions. In another point of view in smart spaces a software on a user's computer may behave differently depending on where the user is currently located.

The second is invisibility - according to the vision of Mark Weiser pervasive computing has got to exclude consciousness from the operation. In practice, a reasonable approximation to this ideal is minimized user distraction. If a *pervasive computing* environment continuously meets user expectations and rarely presents a user with surprises it allows interaction nearly on subconscious level.

The third is local scalability - as the size of a smart space grows the number of participating devices and hence the number of interactions between the user and the surrounding entities increase. This can lead to lack of bandwidth, more power consumption and inconvenience for the users. The presence of multiple users will further complicate the problem. Previous works on scalability ignored physical distance - a web server should handle as many clients as possible regardless of whether they are located next door or across the country. In *pervasive computing* the number of interactions should decrease if the distance between the user and the smart space increases otherwise the system will be overwhelmed with interactions of little relevance. It is also important to allow users to send requests to a smart space from thousands of kilometers away.

The last one is the ability of masking areas with uneven conditions. The penetration of *ubiquitous computing* is dependent of many non-technical factors like organizational structure, economics and business models. Uniform penetration, if ever achieved, is many years or decades away. Hence the difference between the "smartness" of different areas will be huge. There surely will be places e.g. offices and buildings with more modern



equipment than others. These differences can be jarring to a user, which contradicts the goal of creating an invisible computing infrastructure. One way to reduce the amount of variation seen by a user is to have his or her personal computing space for "dumb" environments. As a trivial example, a system that is capable of disconnected operation is able to mask the absence of wireless coverage in its environment. The main arising problems of development and realization of a pervasive system are:

- tracking user intentions;
- exploiting wired infrastructure to relieve mobile devices;
- adaptation strategies: applications must adapt to the needs of the system and the system must be able to adapt to the needs of the applications as well (QoS);
- high level energy management, physical and performance planning;
- context awareness;
- balance between proactivity and invisibility;
- security and authentication.

### 1.3 A Vision Scenario

After having known the base idea and research thrusts of *ubiquitous computing* I draft a vision scenario for better understand of the opportunities resided in such kind of human centered technology.

Fred is in his office, preparing for a meeting at which he will give a presentation and a software demonstration. The meeting room is a ten-minute walk across campus. It is time to leave, but Fred is not quite ready. He grabs his wireless handheld computer and walks out of the door. The ubiquitous system transfers the state of his work from his desktop to his handheld, and allows him to make his final edits using voice commands during his walk. The system infers where Fred is going from his calendar and the campus location tracking service. It downloads the presentation and the demonstration software to the projection

computer, and warms up the projector. Fred finishes his edits just before he enters the meeting room. All of the devices in the meeting room could be Blown-up enabled. The projector offers a projection service the laptop has a keyboard service while Fred's PDA runs a slide show service. The slide show application needs a projection service and a keyboard service to run. When Fred arrives into the range of the projector and the laptop, he selects the two services mentioned above and connects them to his PDA's slide show service. So Fred displays his slides using the projector while his work remains on his PDA, and uses the keyboard of the laptop to switch the slides. As the presentation proceeds, Fred is about to display a slide with highly sensitive budget information. The system senses that this might be a mistake: the room's face detection and recognition capability indicates that there are some unfamiliar faces present. It therefore warns Fred. Realizing that perceptual technology is right, Fred skips the slide. He moves on to other topics and ends on a high note, leaving the audience impressed by his polished presentation. Figure: 1.1.

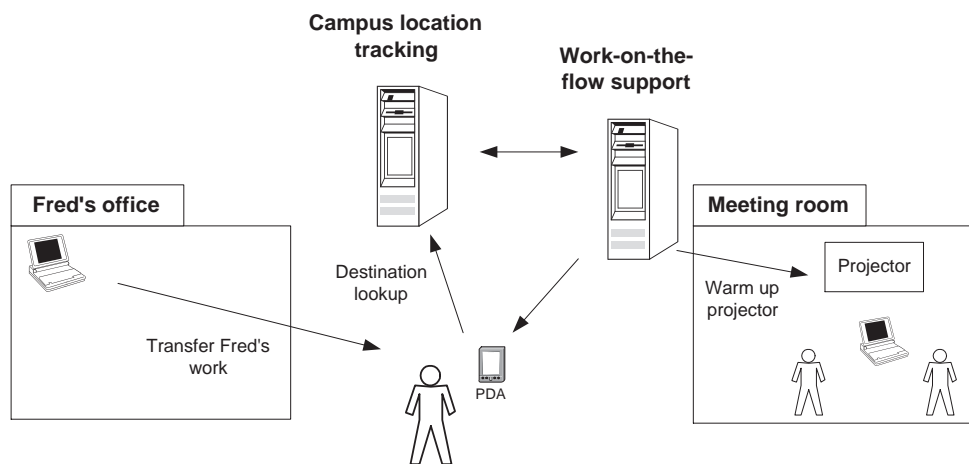


Figure 1.1: Fred's presentation

This vision scenario comes from the researchers of Aura, one of the projects that has been destined to develop a pervasive system. The scenario describes an inter-campus information and collaboration system that aims people working time efficiently. As it can be seen the Blown-up concept could be fit into the example that shows where Blown-up could be used in a ubiquitous vision. However Blown-up needs more user interaction than a pervasive

system should afford, next generation user interfaces could also minimize Blown-up's user distraction.

*Ubiquitous computing* will be a fertile ground for research in the next decades. A lot of new scientific results are needed in many areas, even in those that are not closely related to computer systems, if they are intended to transform a dream to reality. These areas include human-computer interactions (specifically focusing on the variety of interfaces and human-centered hardware design), software agents (with paying attention to high-level proactive behavior) and artificial intelligence (concentrating on decision-making and planning). The capabilities originating from these areas should be integrated into the future systems that are able to fulfill the four major requirements mentioned earlier in this document. So, *pervasive computing* comes to existence as an integration of results achieved in a number of separate fields of science.

## 1.4 Structure of the Thesis

In the last few years many project has been started to make this computing dream or part of the dream into reality. I am going to discuss two of them in the next chapter: the Oxygen and Portolano project. I describe their visions, thoughts, their realization ideas, and then, through these I point to what system support is needed to build applications over a pervasive environment.

In the third chapter I introduce the Blown-up middleware. I present its goals, the architecture that makes realizable the goals. I will also describe the protocol stack in a simplified manner. As my task was on focusing Blown-up applications I avoided writing down the unnecessary details that are not important in terms of application developing. At the end of the chapter application programming interface will be described.

The fourth chapter is about the applications and sessions I have implemented. I introduce the user applications that can form Blown-up sessions, and the control application that can set up and tear down these sessions. I also summarize the advantages of BUMP in terms of application developing.

In the fifth chapter I summarize my experience of Blown-up. I write down those features that needs implementation into BUMP, and those missing features that should be designed to make Blown-up more useful. I also describe the possible future works.

In the sixth chapter I summarize my thesis and the achievements.

## Chapter 2

# Related Work

In the following I will introduce some ongoing projects in the field of *ubiquitous computing*.

### 2.1 Oxygen

Oxygen is a project on human-centered computing started at MIT supported by DARPA and Oxygen Alliance [Oxygen02]. Oxygen enables pervasive, human-centered computing through a combination of specific **user** and **system technologies**. Oxygen's **user technologies** directly address human needs. With the help of *speech and visual technologies* the user can communicate with Oxygen like communicating with a real person and this way they can save a lot of energy and time. *Automaton*, individualized *knowledge access*, and *collaboration technologies* help users perform a wide variety of tasks what they want to do in the ways they like to do them.

*Automation technologies* offer natural, easy-to-use, customizable, and adaptive mechanisms for automating and tuning repetitive information and control tasks. For example, they allow users to create scripts that control devices such as doors or heating systems according to their tastes. *Collaboration technologies* enable the formation of spontaneous collaborative regions that accommodate the needs of highly mobile people and computations. They also provide support for recording and archiving speech and video fragments from meetings, and for linking these fragments to issues, summaries, keywords, and annotations.

*Knowledge access technologies* offer access to information, customized to the needs of people, applications, and software systems. They allow users to access their own knowledge bases, the knowledge bases of friends and associates.

**System technologies** provides location-independent applicability of **user technologies**, they can be used at home, in the office or on the move. The **Oxygen technologies** work together and pay attention to several important themes, figure: 2.1:

- *Distribution and mobility* - for people, resources, and services.
- *Semantic content* - what we mean, not just what we say.
- *Adaptation and change* - essential features of an increasingly dynamic world.
- *Information personalities* - the privacy, security, and form of our individual interactions with Oxygen.

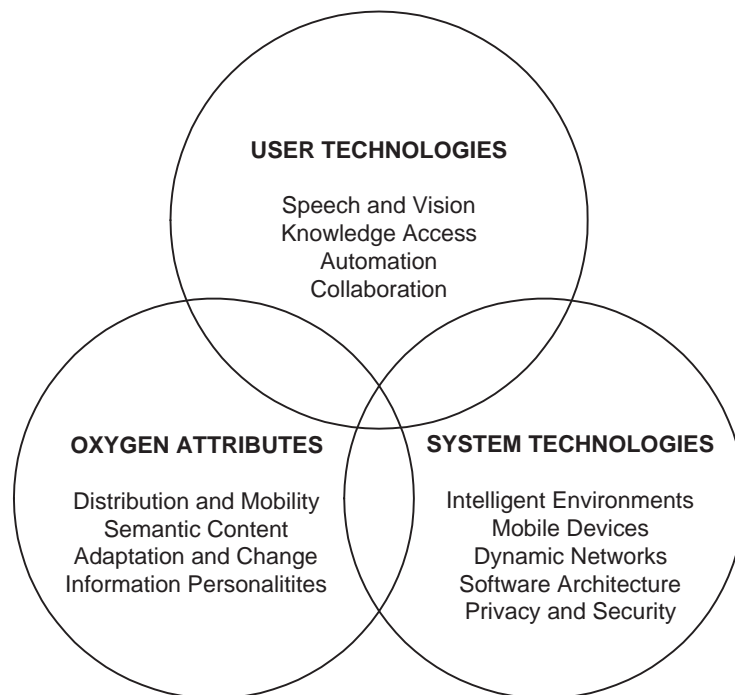


Figure 2.1: Oxygen technologies

People access Oxygen through stationary devices (E21s) embedded in the environment or via portable hand-held devices (H21s). These universally accessible devices supply power

for computation, communication, and perception. Embedded in offices, buildings, homes, and vehicles, E21s enable users to create *intelligent environments* by placing situated entities, often linked to local sensors and actuators, that perform various functions on their behalf, even in their absence. For example, a user can create entities and situate them to monitor and change the temperature of a room. H21s can accept speech and visual input, they can reconfigure themselves to support multiple communication protocols or to perform wide variety of functions. Among other things, H21s can serve as cellular phones, beepers, radios, televisions, geographical positioning systems, cameras, or personal digital assistants, thereby reducing the number of special-purpose gadgets we must carry. To conserve power, they may offload communication and computation onto nearby E21s.

Universally available network connectivity and computational power enable decentralized Oxygen components to perform tasks by communicating and cooperating as much as humans do in organizations. Components can be delegated to find resources, to link them together in useful ways, to monitor their progress, and to respond to change. N21 networks support *dynamically* changing configurations of self-identifying *mobile* and stationary devices. N21s allow people to identify devices and services by how they intend to use them, not just by where they are located. Oxygen networks enable users to access the information and services they need, *securely and privately*, so that people can be comfortable integrating Oxygen into their personal lives. N21s support multiple protocols for low-power communication. The *software architecture* matches current user goals with currently available software services, configuring those services to achieve the desired goals. When necessary, it adapts the resulting configurations to changes in goals, available services, or operating conditions. Thereby, it relieves users of the burden of directing and monitoring the operation of the system as it accomplishes its goals.

As we can see the Oxygen project lines up many cooperative technologies to be integrated in a pervasive environment. Let us see how a business conference would be organized by Oxygen in the scenario follows. Helene calls Ralph in New York from their company's home office in Paris. Ralph's E21, connected to his phone, recognizes Helene's telephone number; it answers in her native French, reports that Ralph is away on vacation, and asks if her call is urgent. The E21's multilingual speech and automation systems, which Ralph has scripted to handle urgent calls from people such as Helene, recognize a previously

agreed word in Helene's reply and transfer the call to Ralph's H21 in his hotel. When Ralph speaks with Helene, he decides to bring George, now at home in London, into the conversation. All three decide to meet next week in Paris. Conversing with their E21s, they ask their automated calendars to compare their schedules and check the availability of flights from New York and London to Paris. Next Tuesday at 11am looks good. All three say "OK", and their automation systems make the necessary reservations.

In the above I drafted what kind of technologies and devices Oxygen focuses on. In the followings I summarize the system technologies and their objects needed to accomplish the proposed goals of Oxygen.

- An integrated visualization and speech system that uses cameras and microphone arrays is required to track a speaker's location and arm position extract the speaker's voice from background noise and respond to a combination of pointing gestures and spoken commands such as "Move that one over here" or "Show me the video on that screen".
- Systems that integrate software services to accomplish user-defined tasks are needed. For example, a smart room equipped with embedded speech, video, and motion detectors automatically records and recalls key meeting events, monitoring and responding to visual and auditory cues that flow naturally from normal interactions among group members.
- A computer-aided design tool is needful that understands simple mechanical devices as they are sketched on whiteboards or tablets. Liberated from mice, menus, and icons, users can draw, simulate, modify, and test design elements in the same way they would with an expert designer.
- Location and resource discovery systems enable users to access computers, printers, and remote services by describing what they want to do rather than by remembering computer-coded addresses. Low-cost ceiling-mounted beacons enable mobile users to determine where they are indoors, without having to reveal their location. These integrated systems respond to user commands such as "Print this picture on the nearest color printer".



- A secure, self-configuring, decentralized wireless network is required that enables mobile users to communicate spontaneously using handheld devices and to share information with one another, utilizing multiple network protocols without requiring additional access points or intervention from service providers.
- Hardware and software architectures that determine and implement the best allocation of resources should be useful for streaming multimedia applications. These architectures optimize computer utilization and the use of power, thereby boosting the performance and lowering the cost of wireless handheld devices that link mobile users to Oxygen networks.

## 2.2 Portolano

For better overview of the technical challenges needed to fulfill the requirements raised by *ubiquitous computing*, I examine the realization ideas of the researchers of Portolano Project at University of Washington [EHAB99].

In a pervasive environment people would be surrounded by computing appliances, but the interactions should occur smoothly and easily. This can be achieved by developing universal **user interface** (UI) for services. Certainly there are many issues in UI evolution, but among these two are especially important.

In a ubiquitous environment there will be services that must be used location independently. No matter people are home or on the move they should be presented a front-end that is appropriate to the capabilities of the device on that they are using the service. For example somebody should be presented with a usable newspaper on a home display or on his PDA's screen or he should be able to complete tasks from many kind of devices. So the first challenge is handling *multiple interfaces*, access points to distributed services. A solution for this problem is a method to allow mobile clients to discover the semantics of any service's UI and present an interface suited to the client's size, shape, abilities, or resource limitations. Research toward this goal is already underway. A first attempt was Interface Description Language (IDLs). IDLs describe abstract UI semantics via a hierarchical set of types. More recently, IDLs have been superseded by a scheme built on top of

the extensible markup language (XML). An other interesting research effort is the VoxML markup language from Motorola. VoxML allows the integration of speech interfaces for interaction with web content through simulated dialogs. Markup languages are an enabling technology, but as good as XML is at describing the semantics and content of a document, it will not be enough. Probably an appropriate solution would be a mobile multi-interface environment. This line of arguments suggest a need for advanced development environments that allow developers to create new user interfaces while re-using much of the code of the back-end of the application.

The second important interface issue is creating *invisible interfaces*. These kind of interfaces would be able to implicitly take their direction from people's behavior instead of giving explicit commands. UI should be made fit so well in an environment that a user will not be aware of interacting with a computing device. One potential part of invisible computing research is context aware computing (CAC). CAC attempts to merge knowledge of the user's task, emotions, location, and attention with other available data such as the time and knowledge about other users. The CAC field is in a quite beginner phase, but already groups are exploring it with projects such as Georgia Tech's CyberDesk and the spatial location work at AT&T Laboratories Cambridge and Xerox PARC. In order to infer intentions data must be fused from a variety of sensors and databases in a timely and efficient manner.

**Distributed services** should be in the core of researches. User must be provided with services to which they can easily relate. For example, somebody may call on several different services to operate on his data e.g. to store his photos, to view them as a web-based photoalbum or to display them somewhere based on conditions of other services. The user does not concern himself with the technical specifications nor the file format conversions.

The current infrastructure-centric focus has led to system architectures that are vertically integrated, not *horizontally layered*. By vertical researchers mean systems that attempt to provide entire solutions to a problem ("take it or leave it"). Traditionally, these solutions suffer from high-cost and inflexibility (e.g. the inability to get information to and from users who do not subscribe to the same service). Although administration and regulation

can be centralized, vertical systems often make it difficult or even impossible for a user to get the subset of services he requires. Furthermore, vertical systems make it difficult to quickly deploy new or alternative services. Researchers argue that horizontal layering is much more appropriate for the mobile networks of the future. If a user desires to migrate to a new service or even use the same service with different options it should be possible for him to easily do so.

Tasks performed by users of a *pervasive computing* environment require the unintentional use of a variety of distributed services. Instead of explicitly communicating with each service, *agents* should perform tasks which are not directly related and are in behalf of users. Technologies and protocols used to implement agents should be able to handle mobile applications in environments with widely distributed data sources and intermittent connectivity. Of particular importance will be an active networking structure allowing the flexible integration of applets and servlets.

Another essential feature of a pervasive environment should be the smooth *integration of new services*. Installations and service set ups should occur seamlessly. Similarly, a new hardware component must be able to setup itself and its connection without the explicit involvement of a network provider. In the current model, significant configuration is required for new devices such as wireless phones. Deploying services effectively also necessitates new distribution and maintenance models. In current systems, users often feel overwhelmed from always having to upgrade their systems with new enhancements, bug fixes, and security patches. Although the user must certainly be kept in the loop about major issues, the day to day maintenance should be the responsibility of the service or subscription provider. This model implies the creation of an architecture supporting dynamic upgrading and hot-swapping of system components. Indeed, many vendors are already addressing this issue. For example, web browsers and multimedia playback tools often come equipped with these autoupdate abilities in order to simplify supporting new codecs and data formats. The challenge will be to implement similar techniques in the mobile domain while spanning heterogeneous hardware and connectivity situations.

**Resource discovery** is the subject of numerous research efforts including the RDP and SLP protocols, Berkeley's Service Discovery Service, Sun Microsystems' JavaSpaces and

Jini, T-Spaces from IBM, Universal Plug and Play from Microsoft, and the HAVi consumer electronics consortium. Each takes a slightly different approach based on the application domain they were intended for. None were designed specifically with mobile networks in mind, so a host of questions should be re-considered in this new context. It is also important that any discovery systems to be implemented should be selfmanaging since both clients and resources (including the lookup registry) are likely to change as devices are disconnected and reconnected. Due to intermittent connections and ad hoc networks, data may have to find services on its own without the assistance of the application that injected them into the network. This necessitates the ability of the networks to execute code in the data packet. This code can call on discovery functions provided on major nodes or, if it finds itself on minor nodes that only route, select the best route to follow to get to those services. To guarantee data safety, this will also require controlled replication of data packets and finite life-times. Acknowledgments from the services that receive the data packets back to the original generators of the data are also problematic as the source may be disconnected or may have moved to a new location. Thus, replies also need to be able to call upon distributed location services help them find their routes or cache data in anticipation of a future connection. While it is important for devices to be able to discover services, they should only be able to use those services for which they have permission. There are protocols that solve network authentication problems like Kerberos or IPsec but the former builds on statically configured centralized servers while the latter does not solve problems of privacy and authentication raised by ubiquitous visions.

Another key issue is the need for **data-centric networks**. Active data bundles should be able to marshal (and pay for) the resources they need to make progress in the network. Data moves from device to device until it reaches the service it is intended for. Though the ideas of ad-hoc networking are valuable, re-think of basic assumptions about network operations and construction of a data-centric network architecture (to distill, name, and locate the data objects that travel within the network) is needed. It is also important for applications to adapt to different network, device and system characteristics. Network infrastructure must be able to inform devices about the network they are using, as well as be able to provide admission control or to guarantee quality of service. Protocols such as RSVP have laid the initial groundwork for effective QoS management in mobile

applications. A data-centric network must be able to manage ubiquitous persistent storage. Sun Microsystem's NFS offers transparent and authenticated access to a global set of files residing on a central server. Unfortunately, this system requires static configuration and has a single point of failure, making it undesirable for mobile applications. Reliability and availability can be increased by using a distributed file system such as the Open Software Foundation's DFS. The Coda file system is a descendant of AFS that is designed specifically for mobile clients. The Bayou project provides mobile clients access to data by using a distributed database approach. Using these projects as starting points, a combination of storage services will need to be designed and integrated into mobile environments. What is really needed to make ubiquitous visions a reality is ubiquitous storage made available to distributed applets running over an ad hoc network. There is still much to study about the consistency semantics in this type of environment in the presence of failures and intermittent connections.

The infrastructure and technology of **distributed computing** also plays an important role in building a ubiquitous environment. Services have to download interfaces to communicate with other services. For example if a service wants to send a picture to an other one, it needs the executable code that provides the appropriate compression format that can be read by the receiver service. Several emerging solutions are to this problem, including Jini and Liquid Software are based on Java language. In these models, bytecode is downloaded and executed on the client. Using the Java RMI, clients can then use the services of other devices. Other solutions are based on CORBA and Microsoft's COM. Both allow clients to execute code located elsewhere, and provide mechanisms that allow clients to discover an object's interface at runtime. The challenge will be in developing an open standard that incorporates the positive aspects of each.

It is likely that **intermittent connectivity** will be the norm for the foreseeable future due to power, cost, bandwidth, latency, and congestion limitations. In order to achieve invisible, trouble-free connections and disconnections from networks, mobility must be built into protocols. By disconnecting during idle periods, devices will consume less power, and lengthen the time between recharging batteries. Not surprisingly, new mobile protocols are already appearing for the intermittent connection environment. Bluetooth and HomeRF are standards for small area radio frequency (RF) networks in which devices can join and

leave ad hoc networks of devices as necessary. These technologies allow mobile devices to join new local networks to take advantage of local resources, it does not address more complex issues such as hand-overs and signal strength analysis found in cellular protocols. Mobile devices also need to be able to get information about the networks that they join. A range of wireless technologies is needed with different transmission ranges and power requirements to support devices like key-chains and earrings that function indefinitely without recharging but have very limited range to more traditional PDAs that are needed to be connected to the Internet and can be more easily recharged. An other important issue is the scalability of the existing infrastructure. In networking RF is promising, but it suffers from a limited bandwidth per volume defined by its range. Irrespective of the medium, security must be integrated into the protocols to satisfy application requirements.

### 2.3 System Support for Pervasive Environments

In the sections before we have known the challenges and problems of *ubiquitous computing*. Question arises that how can we build applications over such a highly dynamic and distributed environment. Noticeable that some kind of middleware is needed to hide or expose the changes of devices, resources, services, and to handle the communication of diverse applications. In the followings I discuss what are the principles of a middleware that is to support pervasive environments and the three main axes along, existing approaches to building distributed systems, fall short, based on the ideas of *one.world* researchers of Portolano Project [GRIMM et al.01].

First, many existing distributed systems try to hide distribution and, by building on distributed file systems or remote procedure call (RPC) packages, mask remote resources as local resources. This transparency simplifies application development, since accessing a remote resource is just like performing a local operation. However, this transparency also comes at a cost in service quality and failure resilience. By presenting the same interface to local and remote resources, transparency encourages a programming style that ignores the differences between local and remote access, such as network bandwidth, and treats the unavailability of a resource or a failure as an extreme case. But in an environment where tens of thousands of devices and services come and go, change is inherent and the

unavailability of some resource is a frequent occurrence.

Second, RPC packages and distributed object systems, compose distributed applications through programmatic interfaces. Just like transparent access to remote resources, composition at the interface level simplifies application development. During composition at the interface level more major application components are used, because they directly reference and invoke each other. As a result, it is unnecessarily hard to add new behaviors to an application, because extending a component requires interposing on the interfaces it uses, and is inadequate for large or complex interfaces.

Third, distributed object systems encapsulate both data and functionality within a single abstraction, namely objects. However, by encapsulating data behind an object's interface, objects limit how data can be used and complicate the sharing, searching, and filtering of data. In contrast, relational databases define a common data model that is separate from behaviors and thus make it easy to use the same data for different and new applications. Furthermore, objects as an encapsulation mechanism are based on the assumption that data layout changes more frequently than an object's interface, an assumption that may be less valid for a global distributed computing environment.

Not all distributed systems are based on extensions of single-node programming methodologies. Notably, the World Wide Web does not rely on programmatic interfaces and does not encapsulate data and functionality. It is built on only two basic operations, GET and POST, and the exchange of passive, semi-structured data. Furthermore, the narrowness of its operations and the uniformity of its data model have made practical to support the World Wide Web across a huge variety of devices and to add new services, such as caching, content transformation, and content distribution.

However, from a pervasive computing perspective the World Wide Web also suffers from three significant limitations. First, it requires connected operation for any use other than reading static pages. Second, it places the burden of adapting to change on users, for example, by making them reload a page when a server is unavailable. Finally, it does not seem to accommodate emerging technologies that are clearly useful for building adaptable applications, such as mobile code and service discovery.

This raises the question of how to structure systems support for pervasive applications. On one side, extending single-node programming models to distributed systems leads to the shortcomings discussed above. On the other side, the World Wide Web avoids several of the shortcomings but is too limited for pervasive computing. To provide a better alternative, three principles are to be identified that should guide the design of a system's framework for pervasive computing.

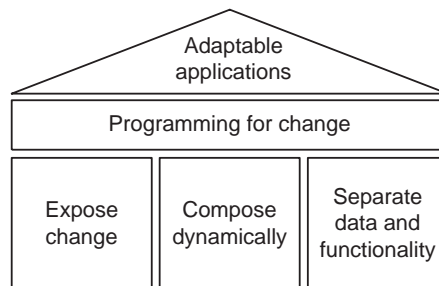


Figure 2.2: System support principles for pervasive applications [GRIMM et al.01]

**Expose change.** Systems should expose change, including failures, rather than hide distribution, so that applications can implement their own strategies for handling changes. Event-based notification or callbacks are examples of suitable mechanisms. At the same time, systems need to provide primitives that simplify the task of adequately re-acting to change. Examples for such primitives include "checkpoint" and "restore" to simplify failure recovery, "move to a remote node" to follow a user as he moves through the physical world, and "find matching resource" to discover suitable services on the network.

**Compose dynamically.** Systems should make it easy to compose and expand applications and services at runtime. In particular, interposing on a component's interactions with other components or the outside world must be simple. Such features make it possible to dynamically change the behavior of an application or add new behaviors without changing the application itself. This is particularly useful for complex and reusable behaviors, such as replicating an application's data or deciding when to migrate an application.

**Separate data and functionality.** Systems need to provide a clean separation between data and functionality, so that they can be managed separately and so that they can evolve independently. The separation is especially important for services that search, filter, or



translate large amounts of data. At the same time, data and functionality depend on each other, for example, when migrating our chat application and the music it is currently broadcasting. Systems thus need to include the ability to group data and functionality but must make them accessible independently. Figure: 2.2.

Common to all three principles is the realization that systems cannot automatically decide how to react to change, because there are too many alternatives. At the same time, a system architecture whose design follows the three principles provides considerable support for dealing with change. Exposing change helps with identifying and reacting to changes in devices and the network. Dynamic composition helps with changes in application features and behaviors. Finally, separating data and functionality helps with changes in data formats and implementation. Given a system that follows these principles, application developers can focus on making applications adaptable instead of creating necessary system support.

## 2.4 Problem Statement

As it can be seen the realization of *pervasive* or *ubiquitous computing* raises a lot of computer technology problems. There are solutions for some problem field but most of these suit the raised claims only in part. The greatest difficulty could be caused by the smooth integration of developed systems that realizes *pervasive computing*. The earlier mentioned research groups have significant results, they can show off working test systems although these are yet quite far from Mark Weiser's vision.

In my thesis I present the Blown-up system, which is a distributed service access technology. Since the technologies, that aims the realization of *pervasive computing*, also focus on distributed services (section 2.2), Blown-up could be fit into these technologies. Blown-up also makes application development easier on devices with limited capabilities and on small ranged wireless networks. Blown-up functions as a middleware and creates system support for personal area network applications.

My goal was to implement the first Blown-up applications (section 4.1), thus present the advantageous properties of the system. I designed the implemented applications to

emphasize the special abilities of Blown-up. My second goal (section 4.2) was to make the applications usable in sessions. To realize this, I implemented a control application, which helps in establishing sessions. By the aid of the implemented applications the advantages of Blown-up can be appreciated not only from developer aspect but from user aspect as well.

## Chapter 3

# The Blown-up Middleware

*Blown-up* concept aims to support distributed services over a personal area network. The system supposes that Blown-up enabled devices form an ad hoc network and hence they are able to communicate with each other. An ad hoc network is not a computing environment by itself, it is only a set of communicating devices. Blown-up system is destined to establish a computing environment, a PAN by connecting network applications or services to each other. At each device all Blown-up enabled network service appear as local, making programmers' work easier. The system relieves application developers of the burden of monitoring such a highly dynamic ad hoc network thus creates system support for personal area network applications.

As mentioned above, Blown-up system associates applications to each other. An *application* in Blown-up can provide software services to other network devices. For example, a device with higher processing performance can run a software service that generates encryption keys for safe communication of devices with lower performance, or may run entertaining services like audio or video decoding. Another example might be a Blown-up compatible game service.

Hardware services can also be offered in Blown-up through software services. For example, a device can offer its storage capacity or its high performance processor by running computing-intensive applications. A device could also offer for example its Ethernet network access through a suitable Blown-up service. Upon this consideration the notion of

*peripheral devices* includes also internal peripheral devices of a computer. This sort of concept implies treating peripheral devices and applications uniformly as *services*.

In a Blown-up network there is a special application called *control application*. Control applications have the permission to connect the offered services thus creating set of connections called *session*. A session is always related to the control application that established it. Blown-up users should build up their personal sessions by the aid of a control application.

*Devices* run the Blown-up services so devices are one level up in Blown-up hierarchy. They are differentiated by Blown-up addresses. Any device that offer Blown-up compatible services can join the system. Devices that support running only one type of service can also be included into the network.

To develop applications onto this architecture system functions must be accessible for programmers. The application programming interface (API) serves this purpose, which enables developers to make their programs Blown-up compatible and so making creation of applications easier in an ad hoc network.

### 3.1 Goals and Assumptions

The following scenario presents the opportunities of *Blown-up*.

Ralph arrives to his hotel room in Paris. On the way to the hotel he added the finishing touches to his meeting presentation, but a little work is still left. As so far he worked on his PDA he got tired of it due to its small touchpad and display. In the hotel room he starts the *Blown-up* control application and look up the services offered by devices in the hotel room. He orders the control application to set up connection to the wide screen display of the room and to the wireless keyboard placed next to the bed. The controller sets up the connections so Ralph can finish his work while relaxing (Figure 3.1).

After he finished the presentation he walks to the hotel conference room in which he agreed with Helene to meet. He finds Helene there but the third person, George is late. He notices by the service discovery function that Helene has a game station in her PAN which runs

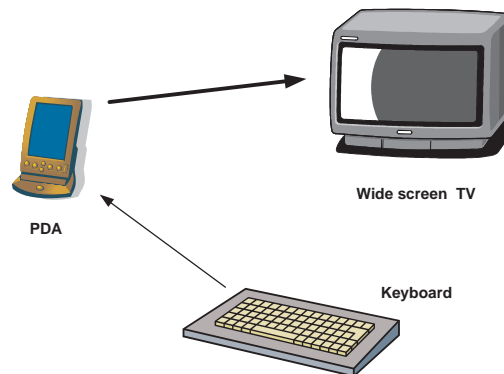


Figure 3.1: Working easy

a chess game. He asks Helene to let him play with the chess game until George arrives. Helene allows Ralph to use the chess game so he connects - using the control application - the display output of the chess game to his PDA's display and the keyboard input of the chess game to the wireless keyboard located in the room. From this time on the game station belongs to Ralph's PAN. In this case the game software runs on the game station, the wireless keyboard serves as data input device, the display output of the game application appears on Ralph's PDA. Ralph also would like to use the projector but Helene still works on it. So when he sets up the connections he also links the display output of the chess game to the projector, and when Helene finishes working and gives up the usage of the projector he can easily switch to it (Figure 3.2).

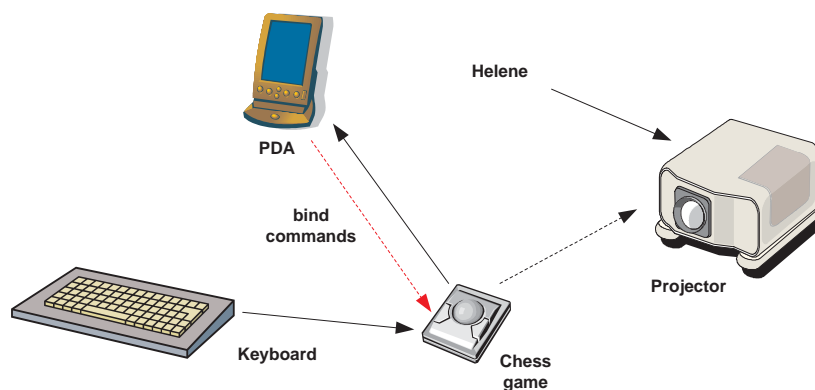


Figure 3.2: Time to play!

Later Ralph gets bored with the chess, but George has not arrived yet. He decides to listen

to music so he redirects the graphical user interface of the mp3 player which can be found on the conference table to his PDA. He would like to listen to the mp3 songs from the conference room fileserver so he connects the server and the mp3 player. The only thing he needs is a headphone which he connects to the output of the mp3 player and now he is ready to start listening to music.

After a while Helene wants to play chess and asks Ralph to join the game. Ralph sets up the previous game connection system including Helene's cellular phone keyboard on which Helene wishes to play. The display output of the chess game is now linked to the projector, the first keyboard input is connected to the wireless keyboard used by Ralph, the second keyboard input is connected to Helene's cellular phone keyboard. Now they are ready to play against each other (Figure 3.3). Fortunately a few minutes later George arrives so they finish playing chess and start the conference.

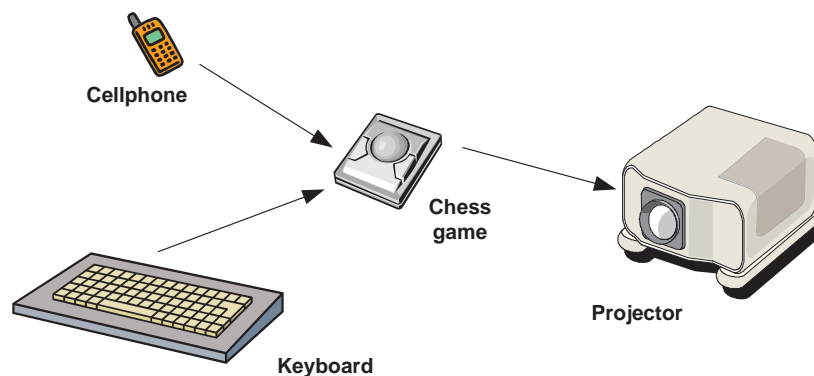


Figure 3.3: Play against each other

An interesting situation arises when Ralph binds the projector and PDA's display together to the display output of the chess game while the projector is already in use by Helene. Question arises that which picture will appear on the projector: Ralph's chess or Helene's work? The explanation of this will be discussed in the later sections under headword *changing focus*.

Three crucial requirement can be noticed from this scenario that such a system should meet. First, from the aspect of usability connection configuration set-ups must be manageable fast and easy - nobody wants to bother with time consuming initialization of sessions. If it

is too complicated task to set up connections people will not use the system. Thus minimal user interaction must be sufficient and a few useful automatism must be implemented into the system (control application GUI, section 4.2.2). Second, secure information exchange must be provided. An unknown person can not be authorized to intercept our data as he gets into the range of our personal area network. Third requirement that can be expected is, that Blown-up must provide standardized interfaces for the communication of same type of services. Without standardization services could not determine the exact meaning of network data.

### 3.2 System Architecture and Protocol Stack

The Blown-up system was designed to be a middleware under applications and over operating system or hardware (Figure 3.4). The Blown-up Micronet Protocol (BUMP) can operate reaching the hardware directly on less powerful devices, without operating system, or as part of or over operating system on more powerful devices.

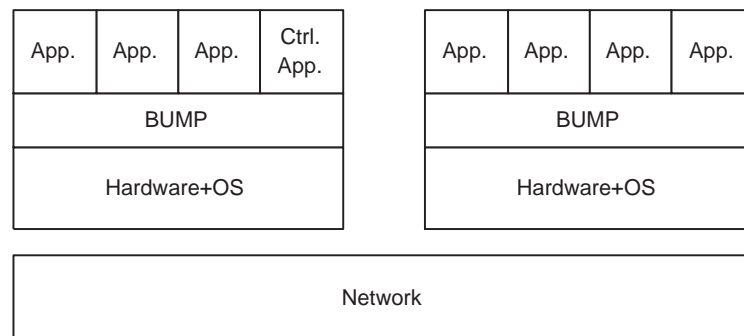


Figure 3.4: System architecture [TDK2002]

By the aid of Blown-up, services offered by applications in the network can be seen and used by applications running on different devices as if they were offered on the same device. The chess game presented above can see the projector in the room as if it were its own resource.

In the Blown-up system applications running on each device appear by their inputs and outputs. To the analogy of the usual naming used in the field of Integrated Circuits (IC)

the inputs and outputs of applications are called *pins*. Each application can be used via its pins. If one application has a free output pin - e.g. a keyboard output - then it can be connected to another application which has the same type of input pin; if one has a free input pin - e.g. a display input - then it makes its pin available for other applications. In this approach Blown-up think in not just applications but also in peripheral devices: a program is treated as a structured set of inputs and outputs. It accepts data on its inputs, puts data onto its outputs - its exact operation remain hidden from other applications. These applications are connected to each other with the aid of the control application.

Connections are defined to be point-to-point. A connection between two pin is called *channel*. Channels implement simplex dataflow (possibly with acknowledgments returned, Figure 3.5). In many cases more than one channel is needed to be established to use a service offered by an application (in section 3.1 the chess game needs a keyboard input and a display output at the same time). Such a set of channels is called a session.

One output pin can be connected to more input pins as well as one input pin can be connected to more output pins at the same time, but only one of them is used at any given time. This channel has the *focus*. Focus can be assigned to another channel by the action called *changing focus*. Focus change is similar to "ALT-TAB" window focus changes used by various operating systems: user focuses on one application, while all other running applications are in the background. Referring to the scenario again, focus change is needed, when Ralph switches the display output of the chess game from the PDA to the projector.

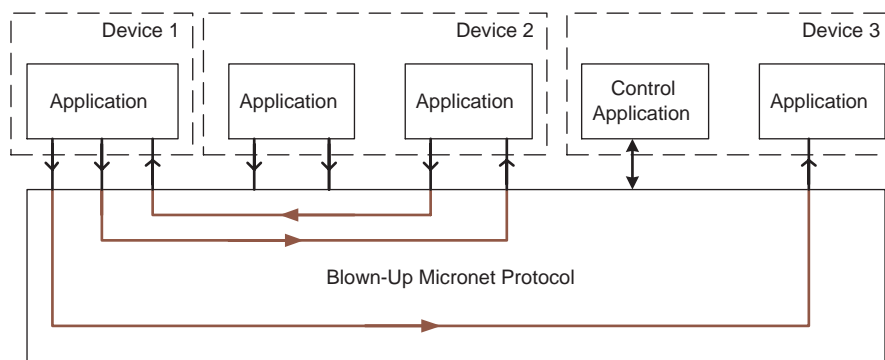


Figure 3.5: World seen by applications [TDK2002]

A great number of programs and hardware devices can offer lots of in- and output. Cer-



tainly a keyboard input should be linked to a keyboard output. Linking an output of a random number generator, for example, to a display input does not make any sense, so Blown-up uses typecheck before establishing connections.

The presence of a control entity, which organizes connections is necessary in the network. In terms of Blown-up there are only two types of application: *user* and *control application*. However we can further differentiate two more type of control applications. The first one is responsible for establishing and monitoring sessions. This type of control applications can be run on a PDA, which has probable enough capacity to supervise a whole PAN. The other type of control applications are those user applications that needs control attribute to operate. This latter type of control application can use the same registering functions as the former control application so Blown-up does not differentiate them, however there is a need for this second type. The reason why this type of applications are needed and the problems coming from not differentiating two type of control applications, will be discussed in section 4.2 and chapter 5.

Many types of computational devices can cooperate in the Blown-up system. Desktop PCs, servers, notebooks, cellular phones, or even some special hardware designed for supporting some special application. The latter can be some appliance, playstation, other software station, or standalone computer peripheral devices (e.g. a mouse or keyboard) which has radio network interface.

The goal of Blown-up Micronet Protocol is to support network communication by the usage of its application programming interface. The system presents opportunity for application developers to create programs easily on distributed ad hoc type PANs. By the usage of API functions implemented in language C++, programmers can prepare their applications for BUMP support.

### 3.2.1 Overview of the Protocol Stack

The Blown-up system uses the Blown-up Micronet Protocol, which consists of three layers: a transport, a network, and an adaptation layer. As BUMP is a middleware, applications run over, and layers for information transfer operate under BUMP layers. As we can see on

figure 3.6 the protocol stack is divided into two more parts, a **user** and a **control plane** (BUMP controller - BUMPC). The applications use the user plane for real communication, the control plane to manage connections.

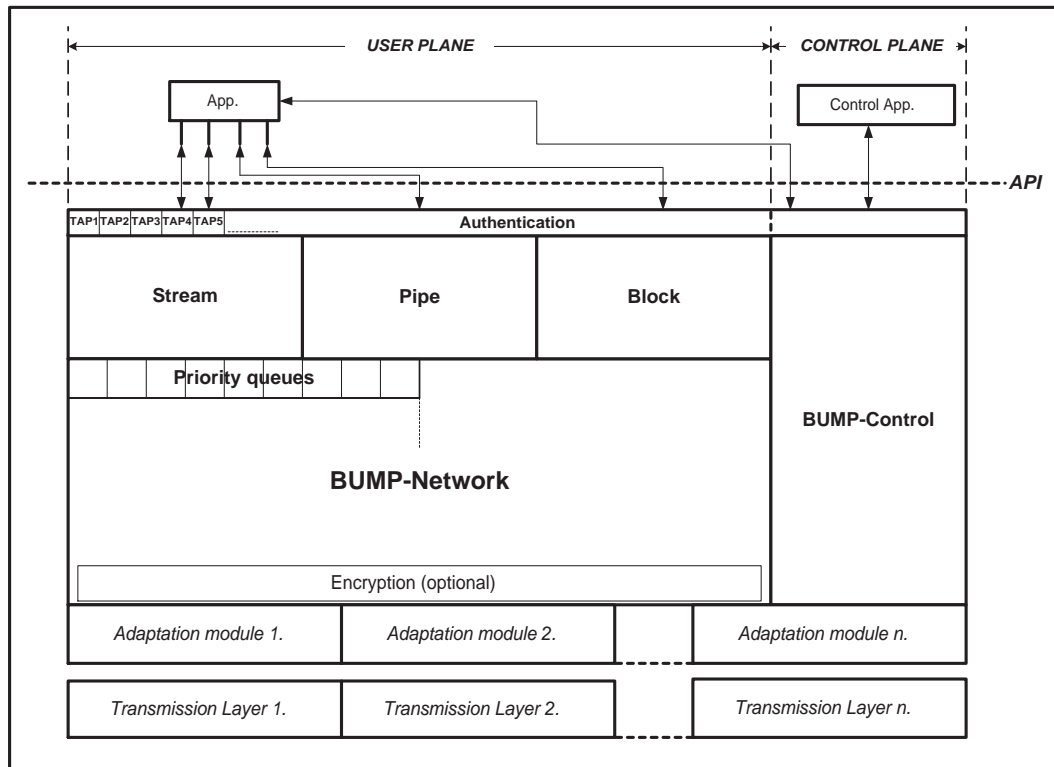


Figure 3.6: BUMP layer structure [TDK2002]

Applications are connected through user plane to the modular BUMP transport layer by their pins. Pin connections are point-to-point by definition, but attaching a special duplicator application makes point-to-multipoint connections possible (for example an application which has one video input and multiple video outputs). All of the pins use a given type of transport module. Pins may send data through these modules to or may receive information from other applications. If a user wants to connect applications with the aid of a control application or modify the existing connections he can send commands to the control module (BUMPC) located on the control plane.

### 3.2.2 User Plane

#### Transport Layer

The role of transport layer is to forward the data packets to the same layer of the device on the other side of a connection, using the rules specified in the given transport layer, and to find the right order of data packets received. Transport types are differentiated according to dataflow characteristics (transaction-based, reliability, flowcontrol, maximum transfer unit etc.). All applications, which want to communicate have one or more pins connected to a transport layer module through an access point (Transport Access Point - TAP). Applications should define, which module their pin should be connected to, depending on the type of data channel they wish to register. So far Blown-up Micronet Protocol defines three type of modules: stream, pipe or block. The parameters of the transport modules can be seen in the table below.

	Stream	Pipe	Block
data unit	lower than 100 bytes	lower than 100 bytes	1 block
type	flow	transaction	transaction
reliability	not reliable	reliable	reliable
flow control	yes	no	yes
data rate	low	low	high and bursty

Table 3.1: Stream, pipe and block transfer module types [TDK2002]

The pins of applications are registered in the transport layer. To every pin the following attributes are assigned:

- the **owner**, that gives the application the pin is registered to;
- the **priority** in case of writable pins. This attribute plays role in sending data;
- the **direction**, which gives the data flow direction of the pin (input or output);
- and a **boolean variable**, which shows that a pin is in disabled or enabled state.

A pin can be in disabled state in two cases. A pin is disabled if it is an output pin and it is out of use. This case can occur when an application may be sending data continuously

onto its output pin, but no-one uses it, so there is no point in forwarding the data into the network. The other case when a pin can be in disabled state when it is an output pin, it is connected and it does not have the focus. This case occurred in the scenario in section 3.1, when Ralph connected the display output of the chess game to the projector despite Helene was still using it. Helene's work appears on the display (the focus was on Helene's channel), so Blown-up can not permit the chess game's display output pin to send data.

The only transport module which has been already designed but not yet implemented into Blown-up Micronet Protocol uses pipe characteristic data transmission. This kind of transport applies a protocol with selective acknowledgments and congestion window that can be used efficiently in case of lossy channels. Applying pipe transport, the receiver does not drop a frame just because it does not received the frames in the right order. It stores the received frames and waits until it receives the missing ones.

### Network Layer

The task of **BUMP-Network** is to handle and supervise established connections. This layer registers pins of local applications and pins of remote (or local in special case) applications connected by a channel (TAP - TAP couples). The BUMP-Network accepts messages from upper layers, forwards them to the BUMP-Network of the other device laying on the other side of the connection, through **adaptation layer**, then passes the data up to the transport layer. It contains message queues for prioritized dataflow in the direction of adaptation layer and an encryption module to encrypt the data injected into the network. The encryption module has not been designed yet.

### Adaptation Layer

The lowest layer of Blown-up Micronet Protocol is the adaptation layer, that transforms the messages of BUMP-Network into a form suitable for the transmission layer. BUMP could be used over any kind of network technology as it is MAC layer independent. Currently it is implemented over UDP/IP but it can be attached to any type of protocol by the modification of the adaptation layer. Since IP has many functions which are unnecessary

in a personal network BUMP could be efficiently used directly reaching the MAC layers (for example there is no point in implementing the IP stack into a mouse). By additional adaptation modules BUMP can be built over WaveLan (IEEE 802.11), Bluetooth or other RF technologies.

### 3.2.3 Control Plane

Connections of the pins are built, torn down, and managed by **BUMP controller** in the network. BUMPC is responsible for discovering services offered by network devices. BUMP controller performs the following tasks of the control plane:

- advertising services offered by local applications;
- monitoring and collecting services offered by remote applications, and make this information available for local applications;
- connecting pins of local applications to pins of local or remote applications;
- establishing sessions for local control application or local user application with control attribute, managing it and revoke after usage.

Applications send registering and connection management messages to BUMPC. Control applications are able to manage the connections initiated by themselves. So while a control application is capable to connect remote services without effectively be involved into that connection system, a simple user application do not have the permission to perform this. The authentication module controls these operations above BUMPC: it registers all applications and their permissions applying to the BUMP, and examine program requests by these entries.

#### Registering Applications

Applications in Blown-up system offer at least one output or one input pin. To send or receive data through them, the application has to be registered into the system. The BUMPC has to know the following attributes of a registered application:

- the **name** of the application (AppName);
- the **description** of the application (AppDescr), from which we can learn additional information about the application;
- the **internal identifier** of the application (AppID), which together with the device address globally identifies a service;
- the **characteristic** of the application's pins, which are the followings:
  - **service type of the pin** (PinType), which gives what kind of information it handles (for example 'keyboard'). The type of the pin provides typecheck mechanisms during establishment of connection, avoiding to connect two pins with different types;
  - **dataflow direction** (Input/Output) gives whether a pin is readable or writable. A pin can not be readable and writable at the same time, so two pins have to be used for duplex transmission;
  - the **necessity** of the pin (Optional/Mandatory), to mark pins that must be connected for the service to work. Optional pins should be connected on demand. For example, in section 3.1 there is no point in starting the displayless chess game without connecting it to the PDA's display (or to the projector), thus the display pin of the game application is mandatory;
  - the **priority** in case of writable pins. This attribute plays role in sending data. This pin information is used at the BUMP-Network priority queues;
  - the **capacity** is the maximum number of pins that could be connected to a given pin. A channel (a link between two pins) can be selected with focus change in terms of dataflow;
  - the **type of the transport module** used by a pin (TP-Type);
  - the **internal identifier** of the pin (TAP-ID) which gives the transport service access point of the pin;
  - the **outer identifier** of the pin given by the user (MyPINName), which aims handling and identifying pins user friendly by a simple name string. Referring again to section 3.1 the chess game have to use different names to its pins.

When Ralph sets up the two player chess game session, the name of the two keyboard input of the chess game must be different to help Ralph differentiate the inputs. In case the capacity of the pins is higher than one the two player could be linked to the same input pin so Helene and Ralph could not manage to establish the desired session obviously.

The BUMPC manages a service registry table where these information are located. BUMPC is also informed of applications registered to a BUMPC of another device by beacon messages, which messages only contain the most important information needed to handle an application: the device address, application identifier and the application name. BUMPC checks whether it gets beacons continuously from applications that it already knows. If not, it deletes the service from the list of usable services. The ceasing of beacon messages can be caused by two reason: the application stopped, or the device which runs the service can no longer forward messages to a given BUMPC. The revoke of the service means that the service finished its operation in the network.

### 3.3 Application Programming Interface

The Blown-up Micronet Protocol provides an application programming interface to reach system services. With the aid of the API, application developers can easily make their programs compatible with Blown-up. The functions the API contains can be organized into two subsets: control and user functions. Control functions provide reaching those system services that are necessary for a control application while user functions aim simple user applications.

In the followings I describe the API functions, their attributes and the system services they reach. The API functions can be included by a simple header file. Parameters, return values, and structures used by this functions are described in details in the appendix.

## Control Functions

**bump\_RegisterControlEntity:** this function have to be called upon control application startup. It registers the control entity into the BUMP stack. Without calling this function the control services of BUMP can not be used.

**bump\_GetApplicationsReset:** resets application listing query. Since the next function returns applications one by one, this function have to be called if we want to start the query again from the first application.

**bump\_GetApplications:** this function returns a Blown-up application into the memory area given as a parameter. The size of memory area must be equal with the size of `AppItem` structure. The return value of the function is `NULL` if the returned application item was the last one in the BUMP register, else it returns a positive value.

**bump\_GetPINsReset:** serves similar function as `bump_GetApplicationsReset` but it should be applied before pin queries.

**bump\_GetPINs:** has a similar role as `bump_GetApplications` but it writes a PIN structure into the given memory area.

**bump\_GetInfo:** returns the description of the application given in parameters to a memory area. The maximal length of the memory area also has to be given. The return value of the function is the size of the string that has been written into the memory area.

**bump\_CreateSessionStart:** this function starts a transaction that aims creating a connection system. The return value of the function is a transaction identifier which is to refer the started transaction.

**bump\_BindPINs:** in case of creating sessions we have to identify the involved links by pin pairs. One pin pair is stored into a `PINPair` structure by their device address, application identifier and pin name. This type of structure has to be passed on to the `bump_BindPINs` function completed with the transaction ID it relates. The return value of the function is a positive value on success or zero on failure.

**bump\_CreateSessionEnd:** marks the end of the transaction which has the ID given in its



attribute. The return value is positive value if BUMP managed to build up the whole session.

**bump\_ChangeLocalFocus:** this function changes focus on a pin belonging to an application running on the local device. As appropriate it takes an application ID and a pin name. Return value shows success or failure.

**bump\_ChangeRemoteFocus:** almost the same as the function above, but changes focus on a remote pin. This function comes with the restriction that a control application can change the focus of only those pins that is related to it.

**bump\_UnregisterControlEntity:** should be called upon the end of the operation of control application. It deregisters the control entity.

## User Functions

**bump\_RegisterApplicationStart:** a simple application should call this function to register into BUMP. Returns success or failure.

**bump\_RegisterPIN:** registers a pin for an application. A PIN structure must be passed on to this function. Returns positive value on success or zero on failure.

**bump\_RegisterApplicationEnd:** marks the end of the application registration procedure. Returns success or failure.

**bump\_SendData:** this is a simple data send function. In case of sending data this function must be called giving a pin name, a memory area, and the size which long the **bump\_SendData** should read on. The return value is negative if the pin is not connected else zero if the pin is disabled. If the send succeeds then the function returns the number of the sent bytes.

**bump\_ReadData:** is a simple data read function. The same attributes have to be given as were given at **bump\_SendData**. But we have to give the maximum size of the memory area **bump\_ReadData** should write on. A fourth parameter also has to be set. This last parameter blocks or unblocks the read function. In case of blocking the function does not

return until it can read data from the channel. The return value is the size of data read.

`bump_RevokeService`: should be called upon the end of the operation of the application. It revokes the service from Blown-up system.

## Chapter 4

# Applications for BUMP

In the previous chapter I introduced the two unique features of Blown-up concept. The first one is that Blown-up simplifies application development over ad hoc networks by allowing developers to register resource requests statically, thus, require the presence of certain resources. The other one is we can use distributed services in an ad hoc network with the aid of Blown-up, hence services can form our own personal network. In this chapter I present the benefits of Blown-up programming interface through two Blown-up enabled services: a program, which aims playing mp3 music, and a file service application which is able to handle simple file operations. After I introduced their features and operations I present the main application of the Blown-up concept: a control application and its Graphical User Interface (GUI) that simplifies its usage.

### 4.1 Blown-up User Applications

Blown-up *user applications* are those that offer or request services in a Blown-up enabled network. Blown-up aims helping the collaboration of these services. All kind of services we would like to use needs a server and a client program. Servers offer services maybe attending more clients, while clients use the services. However in terms of Blown-up those applications are special that offer and use services at the same time; these type of programs could be called as hybrid programs. For example the chess game in section 3.1

uses at least one keyboard service and a display service and offers itself as a chess service. In the followings I am going to introduce a simple file access service and an mp3 service. In this case the file server acts as a simple server, the mp3 client is a simple client while the mp3 server is a hybrid application as it is going to translate the file access requests of the mp3 client into a request for the file service. The implemented configuration can be seen on Figure 4.1.

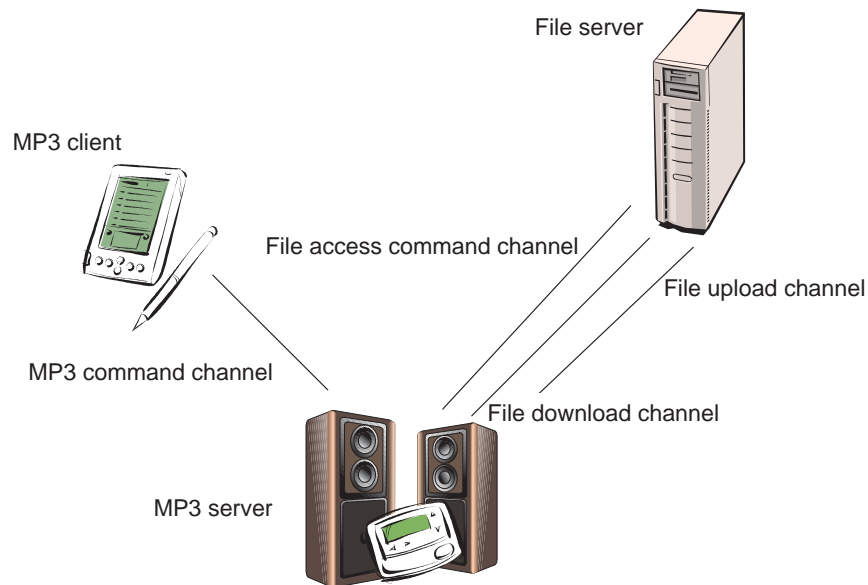


Figure 4.1: Implemented session

The mp3 client has an output pin which injects mp3 playing commands into the channel. The other end of the channel is handled by the mp3 command input pin of the mp3 server. The mp3 server has two other pins. One of them is an output pin which forwards file operation requests. Accordingly a file download pin is also needed to download the files containing mp3 coded music to the mp3 server. However the fourth, file upload pin seems to be redundant it is not that. If we think in a simple file service server-client session, it is certain that it needs an upload function to place data onto the computer that runs the server program. Upon this consideration the file service applications were implemented with an upload pin. Since only mandatory pins were implemented into the current version of BUMP, this upload pin had to be defined as a pin that is necessary to be connected to a corresponding pin, if we want to use the file service application. Hence mp3 service had to be implemented with this mandatory upload pin. In the next sections I will describe

the operation and functions of the file access service client and server.

### 4.1.1 File Access Service

The file access service was created to accomplish the simple file request operations over Blown-up Micronet Protocol. Only the basic file operations were implemented to this service: `get`, `put`, `dir`, `cd`, since the focus was on presenting Blown-up features. The functions of these commands are the followings:

- `cd`: changes and buffers actual directory;
- `get`: downloads a file to the client device from the device that runs the server program. It gets the file from the directory which was buffered by `cd` command;
- `put`: uploads a file from the client device to the server device to the actual directory typed by `cd`;
- `dir`: lists actual directory;

#### Client

As mentioned above the file service client has three pins. Before usage they must be defined and registered to BUMP. The *command pin* is defined as follows:

```
fscpin.TP_Type = 1;
fscpin.PIN_Type = 4;
fscpin.IO = OUTPUT;
fscpin.OM = MANDATORY;
fscpin.Priority = 1;
fscpin.Capacity = 1;
strcpy(fscpin.MyPINName,"Simple file service command output pin");
```

These instructions fill in the memory area of the `UserPIN` structure (A.2) named `fscpin`. Field `TP_Type` means the transport type the pin uses. As only one type of transport layer

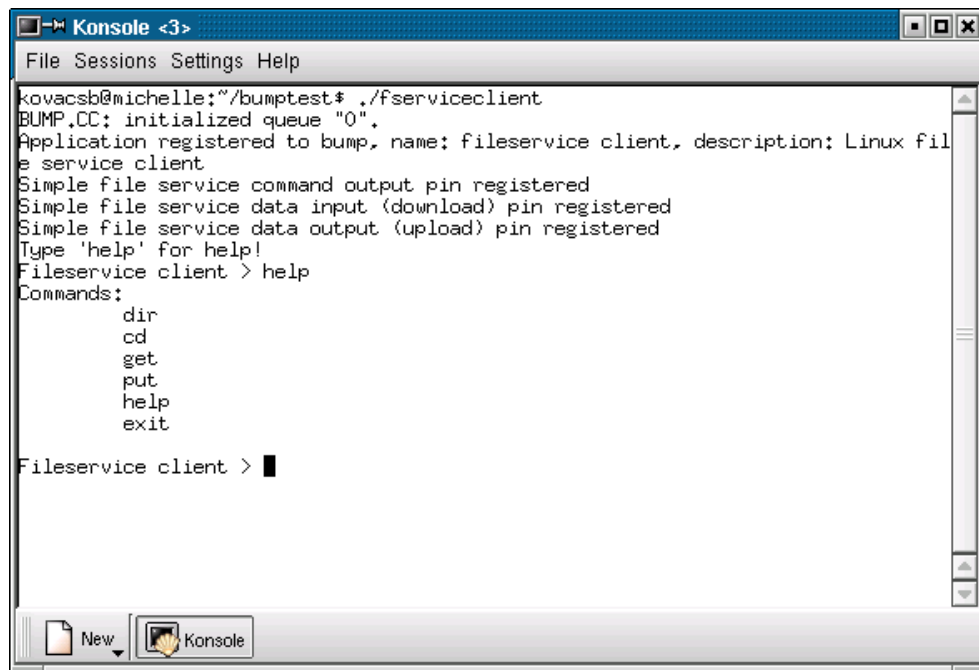
has been implemented into the current version of BUMP middleware, the value 1 is used for all kind of dataflow. **Pin-Type** is a service descriptor. This serves as a service identifier that has to be unique for different type of services. The values of this variable is advisable to be agreed in the future to avoid service conflicts. **IO** defines whether the pin is an input pin or an output pin. **OM** shows the necessity of the pin. It is a boolean variable that can be optional or mandatory. This had to be determined as mandatory in case of every pin due to limitations in current BUMP implementation. **Priority** defines the priority of the data sent on a pin. The lower the value, the higher the priority. **Capacity** shows the number of the pins that can be connected to the given pin. The switches between the connected pins can be managed by focuschange. Finally **MyPINName** is a user friendly string descriptor of the pin.

Apart from the command pin, the file service client has two other pins. The definition of these differs in three parameters from the previous one: the *file download pin* is an input pin, with service type 5 as it accepts file data blocks, while *file upload pin* is an output with the same service type. Certainly the pin name of these pins differs one by one according to their types.

The file access service application is a very simple file access application. The program gets a user command and its parameter (the name of a file) from standard input, processes the command, then sends the asked file through BUMP. Finally it waits for an acknowledgement (ACK) from the file service server on the download channel. In case of changing directory (**cd**) the ACK can be either positive (the directory exists) or negative. Listing the directory (**dir**) can also result in negative ACK in case the given directory does not exist. If **dir** succeeds it returns the directory elements. **Get** downloads a given file to the client device if it exists on the server, while **put** uploads a file to the server device from the client. The command line user interface of the file service client can be seen on Figure 4.2.

## Server

File server aims serving the file access commands from the client. The file server has the same type of pins as the client, except their directions are the opposite ones. Positive or negative acknowledgements are sent back to the client on the download data channel. The



```

Kovacs@Michelle:~/bump$ ./fsserviceclient
BUMP.CC: initialized queue "0".
Application registered to bump, name: fileservice client, description: Linux file
service client
Simple file service command output pin registered
Simple file service data input (download) pin registered
Simple file service data output (upload) pin registered
Type 'help' for help!
Fileservice client > help
Commands:
    dir
    cd
    get
    put
    help
    exit

Fileservice client >

```

Figure 4.2: User interface of the file service client

server executes the following commands on client requests.

In case of a `cd` command the server checks the existence of the directory given as the parameter of the command. If the result of the check procedure was successful it stores the given path and changes the current working directory to it. Every following command is executed in the working path. Of course the parameter could be not only relative, but it could be given with full path, beginning with root: `'/'`.

The `dir` command calls a simple listing function. The listing procedure checks the given path if the `dir` had an argument, if not, it gets the path from the buffer, then gives back the directory elements one by one. The output is sent to the client.

Upon calling the `get` command, the server tries to read and forward the requested file to the client. The full path is read from the current directory stored by the buffer or got from the command line if a full path was given. If the file was found, the server reads the file blocks into a `unsigned char` array followed by an `integer` value that marks end of file. The maximal size of this array is defined by BUMP. The data stored by the `unsigned char` array and the `integer` value is sent to the client.

The `put` command is a reversed `get` as the file is read at the client and written at the server. Certainly the buffered path is also taken into consideration.

### 4.1.2 MP3 Service

The task of the mp3 service is to provide a music player application for Blown-up. I decided to use the X MultiMedia System (XMMS). To make XMMS applicable for the use in a Blown-up enabled network, I implemented client-server applications that are able to communicate with each other through Blown-up. The client sends mp3 playing commands to the server. The server interprets these commands into XMMS commands. Nevertheless Blown-up empowers the mp3 server with a special function described later.

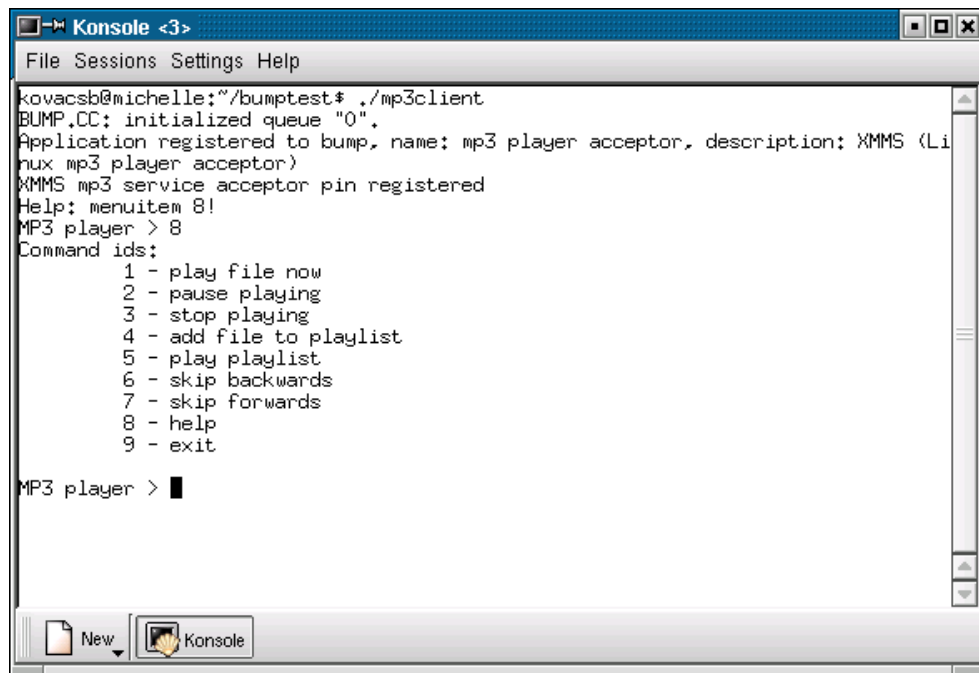
The mp3 client and server are able to interpret the following commands:

- play file now
- pause playing
- stop playing
- add file to playlist
- play playlist
- skip file backwards
- skip file forwards

#### Client

The mp3 service client has only one output pin, named "XMMS mp3 service acceptor pin". This pin sends the user commands to the channel. The commands are sent through the network in a structure that stores the message type and a string. The string contains the name and path of the mp3 coded file on what the user wants to operate. The next figure shows the user interface of the mp3 client.





```
Kovacs@michelle:~/bump$ ./mp3client
BUMP.CC: initialized queue "0".
Application registered to bump, name: mp3 player acceptor, description: XMMS (Linux mp3 player acceptor)
XMMS mp3 service acceptor pin registered
Help: menuitem 8!
MP3 player > 8
Command ids:
  1 - play file now
  2 - pause playing
  3 - stop playing
  4 - add file to playlist
  5 - play playlist
  6 - skip backwards
  7 - skip forwards
  8 - help
  9 - exit

MP3 player > █
```

Figure 4.3: User interface of the mp3 service client

## Server

The mp3 server is the only hybrid application in the session that I implemented. Actually a simplified file service client has been integrated into this application. It has four pins. Three are adopted from the file service client: the command, download, and upload pins. The fourth pin is an "XMMS mp3 service pin" that accepts mp3 player user commands from mp3 service client.

Those mp3 player commands that do not get a filename as attribute (e.g. *stop playing*) are simply translated to system calls and passed up to XMMS. Others like *play* or *add file to playlist* are treated in another way. The only file transfer command this application can apply is *get*. This command is needed to get the user requested mp3 file(s) from a file server to the mp3 server. First the user instructs the mp3 client to play an mp3 coded file, the mp3 server receives the command, then requests the given file from the file server. The mp3 server downloads the music, and calls a system command which starts the XMMS mp3 player with the downloaded file.

### 4.1.3 Summary of BUMP User Plane

The registration of applications and pins into Blown-up can be achieved by those easy-to-use initialization instructions that were presented in 4.1.1. Only a few parameters have to be set that describes service type, dataflow type, direction, etc., and a string to name the pin to be registered. This string is the only one that identifies the pin. Network addresses, port numbers are not needed to be taken into account. The application almost does not see it is over a network.

Before registering the pins of an application, we have to register the application itself. This can be attained by calling a BUMP API function. The name of the application and an application description have to be passed on to the function.

```
if (bump_RegisterApplicationStart("application name",
                                "application description") < 0) {
    fprintf(stderr, "Error: Can not register application!\n");
}
```

After we managed to get over this procedure, the turn is on registering the pins one by one. We have to fill in a `UserPIN` structure (A.2) with those parameters already mentioned. The next example passes on this type of structure called "pin1".

```
if (bump_RegisterPIN(pin1) < 0) {
    fprintf(stderr, "Error: Failed to initialize pin!\n");
}
```

Finally we have to mark the end of the registration procedure to BUMP implying we are not requesting more pins to be registered.

```
if (bump_RegisterApplicationEnd() < 0) {
    fprintf(stderr, "Error: Failed to initialize pins \
                Can not register application!\n");
}
```

As it can be seen all three functions have a success or failure return value. The return value of the first function is lower than zero if BUMP failed to initialize the application. The second function returns success or failure pin by pin while the third returns overall result. In the present implementation of BUMP, `bump_RegisterApplicationEnd` returns zero if one of the `bump_RegisterPIN` returned zero. This comes because of the support of optional pins is not yet implemented.

After the registration procedure sending onto and receiving from the channel is very simple only the following functions have to be called.

```
bump_SendData(pin.MyPINName, &msg, sizeof(struct command_message));  
bump_ReadData(pin.MyPINName, &msg, sizeof(struct command_message), WAIT);
```

The attributes of `bump_SendData` and `bump_ReadData` are quite trivial, they were described in section 3.3. Before the application hangs up its operation over the BUMP it has to call a revoke function to safely remove any record related to it from the protocol stack.

```
bump_RevokeService();
```

These functions above are all that a user application operating over Blown-up has to cope with. They can be easily overviewed and handled, and however the application operates over a network, listening, connecting, accepting clients are not need to be taken into account. Actually we can say that Blown-up makes network programming simple and efficient. Blown-up Micronet Protocol enables the previously described applications to operate anywhere in a Blown-up network, either on the same device or different devices.

In the next sections I describe the BUMP control plane, a control application and its GUI.

## 4.2 Control Application

A control application (CA) is the only application in Blown-up that is able to give control instructions to BUMP. A control application can build up and revoke sessions, list the Blown-up network services or initiate focuschanges. By establishing sessions users can

create their own personal network. That is why usual control applications should be run on more intelligent devices like PDAs or mobile phones from which users can easily create and monitor their sessions. However CAs are to manage connection systems a simple user application can register into BUMP as a CA by calling the control application register functions. These functions are needed if the given application needs some controlling functions to its operation. Let us assume that we run a positioning service on a mouse. We are able to connect more than one acceptor (client) pin to the mouse due to its capacity is higher than one. We connect the mouse to our PC and our laptop by the control application running on the latter. It is quite uncomfortable to hunt up our CA on the laptop every time we want to switch the mouse between the two computer. A more suitable solution is to implement a simple control application into the mouse user application that is able to give focus changing instructions. In terms of Blown-up those applications that call control register functions are treated equally, regardless they are registered as user applications or not.

BUMP does not define how many control application can be present in a Blown-up network, so the number of them is not limited neither in Blown-up network nor on Blown-up devices. Certainly these applications have to be restricted in permission of managing sessions. BUMP permits CAs to access only those sessions that have been established by the given CA and to modify those Blown-up applications that runs on the device of the CA. Problems with control applications' permissions will be discussed in chapter 5.

An important feature of control applications should be to provide an easy and efficient way to set up sessions. Upon this consideration a graphical user interface is advisable to be implemented as the front-end of CAs. Without this, operating a control application could be time consuming and makes the use of Blown-up uncomfortable for users.

In the next two sections I describe the operation and functions of the control application's back-end and front-end.

### 4.2.1 Back-end

The back-end of the control application was designed to be a server for the graphical interface. It realizes the previously described control application functions, stores their results and makes the results available for GUI queries. I implemented the back-end in C++, while I chose TCL/TK to implement the GUI. For their collaboration I decided to use socket communication. The communication method is quite simple. The TCL code gives commands on user interactions to the back-end then the back-end replies or waits for more data (for example at creating sessions). The storage of the Blown-up network information that is available through control plane API (section 3.3) functions, is solved by C++ classes. Classes are appropriate not only to store but to easily make available this information through their member functions.

The back-end provides the following services for the front-end:

- **refresh**: initiates a BUMP query that collects all devices, applications and pins that are available in Blown-up. The result of the query is stored only in the memory of the back-end. The front-end has to apply the following four service to access the refreshed information;
- **get\_devices**: due to considerations will be discussed in section 4.2.2, devices, applications and pins should be looked up one by one. This command initiates a query served from the memory of back-end, and returns the addresses of Blown-up devices for the front-end;
- **get\_applications**: similar as `get_devices` but requests the name, identifier or description of the application on a given device;
- **get\_pins**: similar as `get_applications` but requests the `MyPINName` of given application's pins;
- **get\_pinparams**: requests every parameter of a given pin: pin type, transport type, direction, necessity, priority, capacity, pin name;
- **create**: initiates a session set up. After this command, the pin pairs requested to be connected have to be sent from the front-end. The established session is stored

in the memory of the back-end;

- **show**: lists those sessions that have been created by the control application. It serves back a session identifier and the pin pairs taking part in that session;
- **delete**: disconnects a selected session by session identifier;
- **focus**: changes focus on a selected pin;
- **exit**: exits from control application and unregisters it from BUMP.

In behalf of serving the front-end needed data, two main object have to be stored into the memory of the control application. The first contains pins and the applications they are related to, while the second is for making session information available. The former is solved by class `AppPIN`, the latter by class `ActiveSession`. The efficient storage of these classes is solved by a list template class. A simple UML diagram of the attributes and member functions can be seen on Figure 4.4.

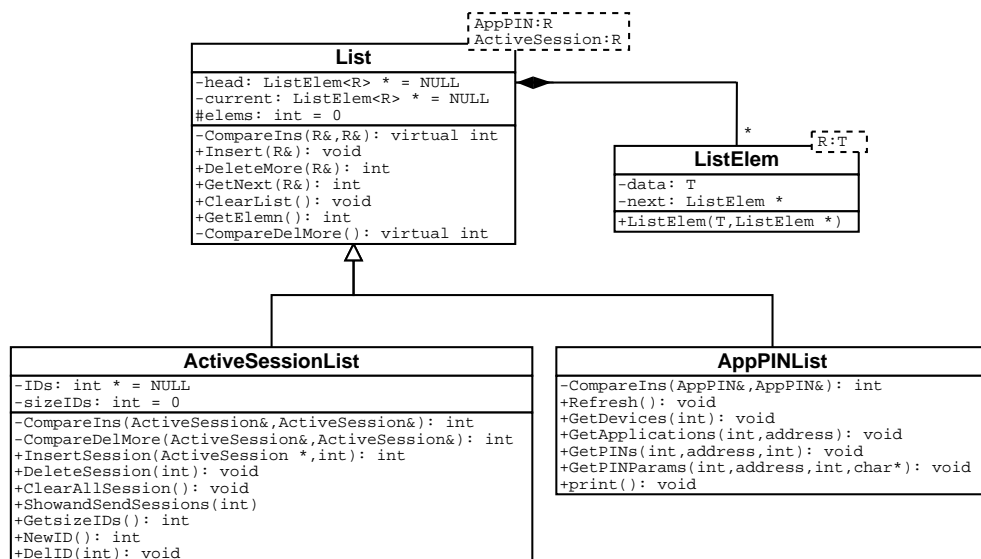


Figure 4.4: UML diagram

## Class `AppPIN`

As mentioned above class `AppPIN` aims storing pins and applications together. This class serves as list element of `AppPINList`. The attributes are the followings:

- **application**: is a `struct AppItem` data structure which stores three main parameters of an application: name, device address and an identifier;
- **app\_description**: contains the description of the application. It is not integrated into the former structure because BUMP returns it by an other function (due to network traffic considerations);
- **pin**: is a `struct UserPIN` data structure which stores a pin and its user-important information.

<b>AppPIN</b>
-application: AppItem
-pin: UserPIN
-app_description: char *
+AppPIN(AppItem, UserPIN)
+AppPIN(AppItem, UserPIN, char *)
+GetApplication(): AppItem
+GetUserPIN(): UserPIN
+GetDescr(char *): void

Figure 4.5: Class AppPIN

Most of the member functions that can be seen on Figure 4.5 are simple attribute returner function (e.g. `GetApplication`). They are able to return the three attributes of the class. As it can be seen there are two different types of constructor for the more attribute flexible creation of class objects.

### Class AppPINList

Class `AppPINList` is the container of `AppPIN`. One pin only occurs once in the list while an application occurs as many times as many pins it has. Opportunity raised during the implementation to store pins and applications in different classes and lists, keeping their relationship together by some foreign key and therefore reduce redundant data. Considering the additional information per item and the number of items that can occur does not implied yet the necessity of more sophisticated database storing algorithms.

Class `AppPINList` is inherited from the template list class, so thus coming into a few useful list manipulating functions like `Insert`, `ClearList`, `GetNext`. The class has one private

compare function which aids inserting into the list sorted. This function is also defined in the template list class as a virtual function. `AppPINList` has five more public member functions that serves GUI requests itemized earlier. Each of them has been designed upon the same consideration. Searching and filtering requested information from database then returning it to the GUI. `GetDevices` selects diverse devices, `GetApplications` filters on a device address and replies the application running on that device, `GetPINs` selects those pins that correspond the given device and application identifier, while `GetPINParams` returns each of the user-important parameters of a given pin.

### Class `ActiveSession`

The relationship between `ActiveSession` and `ActiveSessionList` is the same as the relationship of `AppPIN` and `AppPINList`. The attributes of this class are the followings:

- `activepair`: is a `struct PINPair` structure storing a pin pair by a pair of device address, application identifier, and pin name. These together define a channel;
- `bindID`: is the identifier of the session that `activepair` relates to. Certainly one `bindID` occurs more in the `ActiveSessionList` because more pin pair can take part in one session.
- `ID`: is the unique identifier of the pair. In the present implementation of BUMP, sessions are not able to be modified after establishing them. In case this would be available in the future, `ID` should help in handling pin pairs one by one. In the current CA implementation it does not have any role.

The `ActiveSession` member functions are quite simple. There are two different type of constructor, a destructor, three attribute get function to reach class private members. The latter is important during GUI queries. There are two value adding operator overload functions, one for copying from another `ActiveSession` class, the other is for reading data from a `struct PINPair` into `ActiveSession`. There are also two set functions for setting the `bindID` and `ID` attributes. These are needed, because the session and the unique identifier are got later then the construction of an `ActiveSession` class happens. The



<b>ActiveSession</b>
<pre> -activepair: PINPair -bindID: int = 0 -ID: int = 0 </pre>
<pre> +ActiveSession(int) +ActiveSession(PINPair,int) +GetPINPair(): PINPair +GetbindID(): int +GetID(): int +SetID(int): void +SetbindID(int): void +operator=(ActiveSession&amp;): ActiveSession&amp; +operator=(PINPair&amp;): ActiveSession&amp; </pre>

Figure 4.6: Class ActiveSession

`ActiveSession` class is constructed when the back-end have received the desired session (in pin pairs) from the front-end. However the values of `bindID` and `ID` are got when Blown-up have established the desired session. So a class `ActiveSession` is sent to Blown-up, and then the `bindID` is received. `ID` is drawn later by the back-end to identify a given `ActiveSession` class.

### Class `ActiveSessionList`

Class `ActiveSessionList` serves as the container class of `ActiveSession`, it has been also inherited from the template list class. It has two private compare functions for list operations. One for inserting sorted, the other for getting list elements by unique id-s. There is a constructor and a destructor. There are two additional attributes. The first is `IDs` to store used identifiers while the second `sizeIDs` is for storing the actual size of the `ID` array.

`ActiveSessionList` has three functions that serves front-end requests. `InsertSession` is for establishing user requested sessions, and inserting them to the back-end memory. This function gets a pre-filled `ActiveSession` array. The objects taking part of this array are containing those pin pairs that have been requested by the user and so these pin pairs act as a channel endpoints in user's session. At the beginning of the function all requested pin pairs are asked to be connected by BUMP. Upon success the session is created, `ID`-s and `bind ID`-s are assigned to the corresponding `ActiveSession` object. In case of not succeeding with a given pin pair user could be offered to choose an other pin pair in place

of the failed one, or to accept the reduced connection system. It is very important that the latter can occur only if the pins in the failed pin pair were optional. Since the error handling of BUMP is not fully advanced yet thus there was no point in fully implementing the first feature. However BUMP does not handle optional pins, the second feature could work in the present implementation of the control application, but it automatically accepts the reduced sessions without asking the user.

The last two GUI serving function are `ShowandSendSessions` and `DeleteSession`. The former returns all of the earlier established sessions that has been created by the control application. It returns with session identifiers and pin pairs. The latter deletes and so tears down a selected session by an identifier.

### 4.2.2 Front-end

The front-end or graphical user interface of the control application makes a Blown-up enabled network easily overviewable and handleable. I wrote it in TCL/TK, a platform independent scripting language with graphical extension. I decided to use socket communication for the collaboration of the GUI and the back-end.

TK splits up a window into frames. The main frame of the control application was designed to easily look up Blown-up enabled devices, their applications and the pins of the applications. The additional functions of the CA, like refreshing network status, creating, showing and deleting sessions can be accessed over the menubar.

Listboxes were chosen to handle queries related to Blown-up elements and parameters. As it can be seen on Figure 4.7 four listboxes lays on the mainframe next to each other. The left one stores the devices. This listbox can be filled up with the menubutton *Refresh*, which asks the back-end to refresh its database from BUMP, then gives a *getdevices* command onto the socket. On the figure two Blown-up device appears in the network.

After viewing the devices, we are able to select one of them. The selected item's address is sent back to the back-end after the command *getapplications*. The reply for the command appears on the second listbox from the left by an application description and an application identifier. Certainly only one device's applications show up on this listbox.

The third and the fourth listbox present the application pins. The third one stores output pins, the fourth the input pins. As in the case of applications only one application's pins appear in these two listbox. They are listed by a *getpins* command completed with a device address and an application ID.

In case of clicking onto one of the listed pins we can get the parameters of it. This is solved with a popup window, where all user-important parameters appear. Actually this can be seen on Figure 4.7.

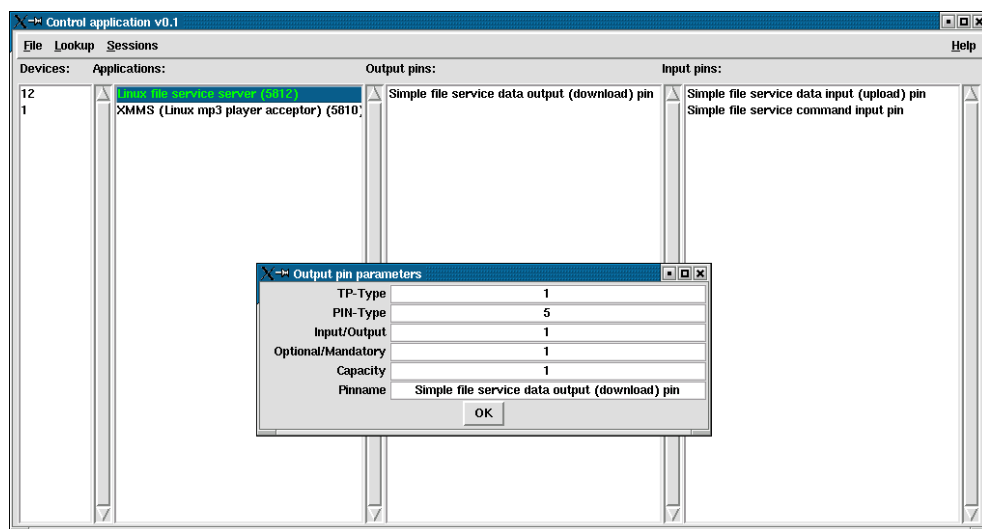


Figure 4.7: Device, application and pin lookup on GUI

After we know the Blown-up services nearby, we can choose one or more applications we want to use. To register and fulfil our request we should click on the menubar, and choose the menubutton *create*. A popup window shows up (Figure 4.8) and the creation of our session can begin. After selecting the first application by control-click, all of the pins appear in the popup window. Each entry contains the device address, application identifier and the name of the pin. The entries show that these pins are needed to be connected to some other application's pins. A label over the entry list make this known to us (Figure 4.8).

Before selecting another application we should check four important parameters of the firstly selected application's pins because these can carry information about the other pin they should be linked to. The one that can help us much is the pin name, which contains descriptive information, but not exact since the attachment of the pins happens

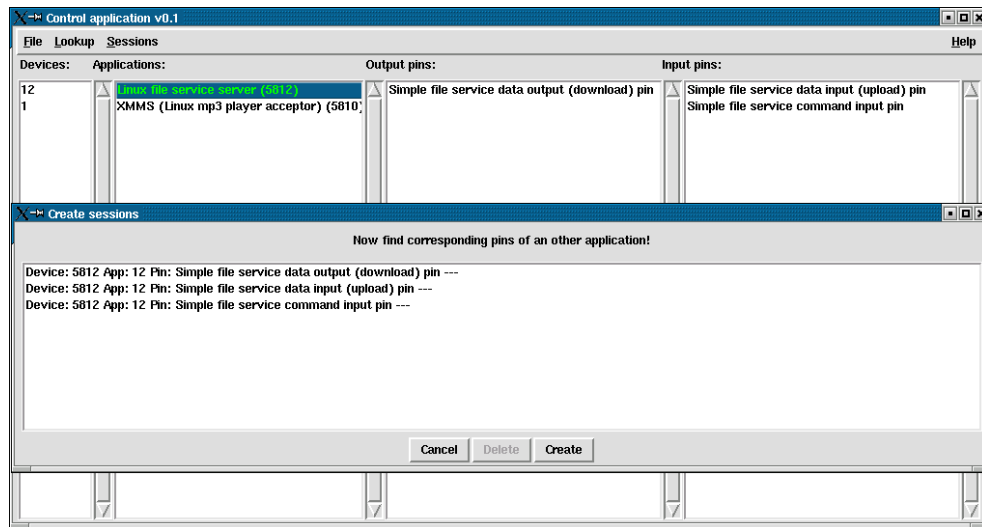


Figure 4.8: Creating a session, step 1

by checking pin type, transport type and direction. So on fast success we should check also these pin parameters by clicking on pin listboxes. After we have known what kind of pin we should search for, we can start browsing again among Blown-up elements. If we have found a corresponding pin, we should control-click on its application. By selecting the create window we can see that the other application's pins also appeared in the window, moreover a pin, that could be paired to a pin of the first application, has been attached to each other (Figure 4.9).

As it can be seen on the Figure 4.9, one pin of the second application could not be paired to any of the pins of the first application. So we have to continue browsing for an appropriate application, to complete our session. After we managed to find one we should control-click on it. In case the control application is able to attach the pins to each other and no more application is requested to take part in the session, we are able to establish the connection system by clicking on the *create* button.

After creating a few session, we can be informed about the operating ones, or we can revoke them from our personal network. This can be achieved by clicking on *show/delete* menubutton of sessions menu. A popup window appears (Figure 4.10) where we can find our useful information and those buttons through which we can revoke our sessions and navigate between them. Clicking on buttons '*«*' or '*»*' step backward or forward among

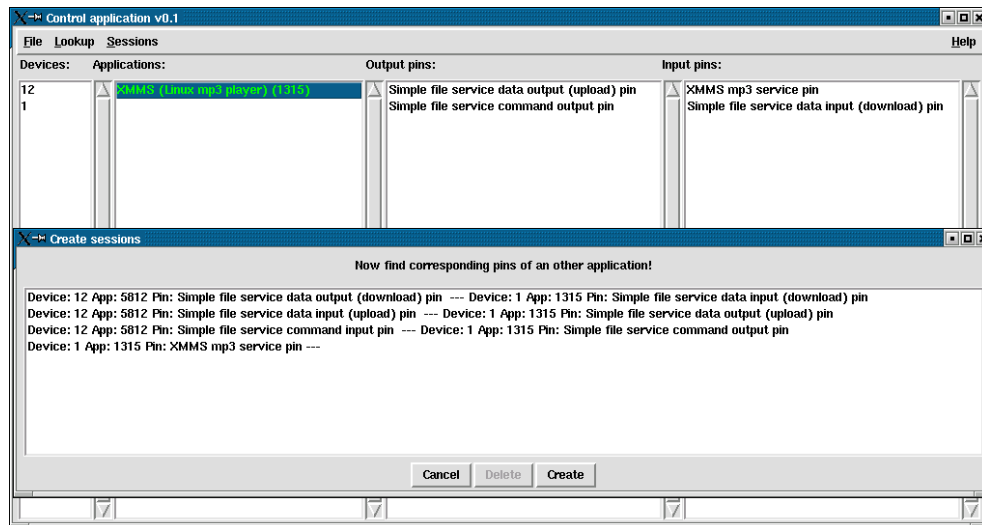


Figure 4.9: Creating a session, step 2

our established sessions reading the actually needed data from the memory of the front-end. The front-end has collected all the available sessions from back-end when we pushed *show/delete* button. If we got bored with one of our sessions we should select it, and click on the *delete* button. In this case the session identifier of this session is sent to the back-end after command *delete* so the control application revokes it from BUMP.

### 4.2.3 Summary of BUMP Control Plane

Some of the control functions of the Blown-up Micronet Protocol 's API operates almost in the same manner as the user functions only their objectives differ. There are also registering functions, one of them registers the other unregisters a control entity.

However a control application has registry functions, those functions that serve creating sessions, are executed in a similar transaction manner as the user application register functions. There is a transaction starter function, a finisher function. Between these functions `bump_bindPINs` have to be called by which more pieces of data elements can be registered. At user applications an application element and its pins can be registered, at control applications a session and its pin pairs should be registered. The return value and error handling happens in the same method.

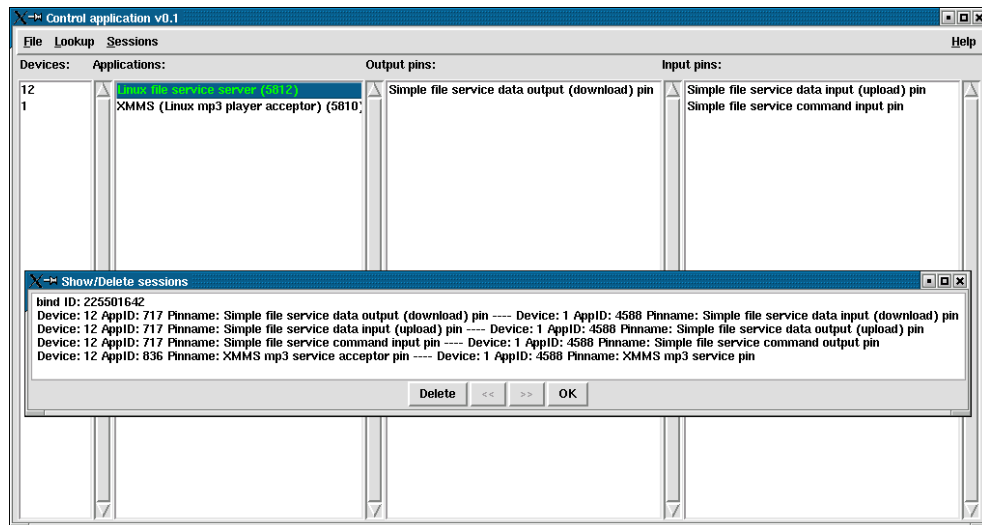


Figure 4.10: Session lookup and delete

The third group of functions serve the query procedures. They work in a very simple manner. There is a reset function, which starts a query from the beginning, and there is another function which returns data elements one by one. The latter should be being called as far as it can give back no data. Blown-up applications and pins can be looked up in this way. The `bump_GetInfo` query function returns application descriptions.

The last two control functions serve changing focuses. One of them changes focus on local device (where the control application runs), while the other on remote device. The main difference between them is that local focuschanges can be executed on every local application, but remote focuschanges are restricted to only those applications that are owned by the control application that wants to change focus.

## Chapter 5

# Analysis of BUMP

In the previous chapters I introduced the Blown-up technology. Blown-up makes easier the work with small devices, expands their capabilities, and also makes easier the application development in ad hoc networks. However Blown-up is a working system, some functions are still missing. These functions are definitely necessary to use Blown-up in such an ad hoc environment it was designed for.

Some of these functions are defined in Blown-up documentation, but have not been implemented yet. One is the availability of *optional pins*. Optional pins are needed if there are additional functions of a service, but optional pins are not necessary to be connected to use the service. So a service can take part in a session without connecting its optional pins. Actually I met this problem, when I implemented the mp3 server. I wanted to use the file service in the mp3 service to download mp3 files but I had to register a redundant upload pin (section 4.1).

Another function is the use of different *transport types*. Since Blown-up was designed to operate over any type of transmission layers, it has to provide different transport methods for applications running over the middleware. As several type of application could use Blown-up hence it must be prepared to provide several type of dataflow.

There are also some properties of Blown-up that have been mentioned in chapter 3 but they have not been designed yet into BUMP. These are *authentication*, *authorization*,

*encryption* and *standardization*. The first three are needed to safely use services over Blown-up. Certainly we can not afford anybody to intercept our data to violate our data integrity or to modify our sessions. Standardization is one of the most important issue about Blown-up. So far when I wrote about a given service (for example keyboard) I supposed that for example a cellular phone's keyboard service can be used by any type of keyboard client. Probably in this case that the key code map of the different type of devices also differs, so a standardized key code map must be agreed. Certainly not only the keyboard but all other service must be standardized, to help Blown-up applications determine the exact meaning of network data.

During the development of the applications I realized three important functions that should be designed and implemented into Blown-up. When I mentioned the type of applications in terms of Blown-up I described user and control applications. However I showed that there can be a third type which uses user and control functions as well, but Blown-up qualifies this, as a control application. Certainly it can happen that a user application with control attribute (mixture application) runs on a device, that also runs another control application by which the user established his sessions. In this case we can not afford the mixture application to modify the established session of the "real" control application. Thus it would be advisable to *differentiate three type of application*. User applications that have no control permissions, user applications with control attribute that have restricted control permissions, and "super user" control applications that have full control permissions. The second feature that should be implemented is *changing focus by channel groups*. Let us assume that we want to use the mp3 server, and download files from two different file server. In this case the mp3 server has to be connected to both file server at the same time, but certainly only one of them is in active state. The mp3 server does not see that it is connected to two file servers. As was mentioned in section 4.1, three pins have to be linked to use a file service (upload, download, command). If the user of the mp3 server wants to switch between the two file server then he has to change focus on all three pins else the other file service server will not work. So there are cases when a focus change on a pin implies focus change on an other pin, hence Blown-up has to solve the focus change in pin groups. The third feature that should be implemented is the opportunity of *modifying sessions by pins*. In the current implementation sessions can be built up and torn down.



In case of a service falls out of a session, the whole session disconnects. Some mechanism should be built into Blown-up that allows the search of a service which is compatible with the failed service. By the aid of this mechanism we could be able to continue the usage of the session with the new service.

## Future Work

Blown-up could be a part of a system that is able to create system support for *pervasive* applications. Certainly not only the above mentioned functions have to be implemented to achieve this. Many special features are needed to make Blown-up appropriate for *pervasive* applications. One of them is the support of mobile code. If we want to use a service for that we does not have the client program, we should be able to download one from the server. For the sake of the solution of this problem, the device, which runs the server program, should store many type of client programs to support the user with an appropriate one.

Another Blown-up future work could be the support of session migration. This means, that the user and his control application from which he initiated his sessions, moves away from his original place, and from the used services as well. As the control application moves, it should be able to search for corresponding services nearby the user's actual position. The control application should be able to rebuild the user's sessions if it manages to find other corresponding services. In other words the session follows the user.

My future work would be to complete my applications and prepare them for safe usage. Since I was focusing on creating sessions that are interesting in terms of Blown-up, some functions are missing from the user and control applications.

Further future work could be to enhance my applications to use the functions of on-developed Blown-up. It is also important to develop more Blown-up applications and sessions that effectively represent Blown-up abilities.

## Chapter 6

# Conclusion

In my thesis I have introduced *pervasive* or in another name *ubiquitous computing*. This field of computer technology set itself an aim to change people's notion of the world of computers. There are still many problems that have to be solved to overcome the difficulties raised by *ubiquitous computing*, but the research is underway. In my thesis I have presented these difficulties, and the ideas of researchers how the difficulties could be solved. I have showed that there is a need for a system that supports the development of *pervasive* applications.

I have introduced the Blown-up distributed service access technology. I have described the basic notions of Blown-up, a scenario that shows the abilities of Blown-up and I have presented its architecture that makes possible the realization of Blown-up's goals. I have described the application programming interface of Blown-up Micronet Protocol that helps in making applications Blown-up compatible.

I have introduced a Blown-up session that can be built up of a mp3 client, a mp3 server and a file server. This session could be called as an mp3 session that allows running the applications of the session on different devices with the aid of Blown-up. I also presented a control application and its graphical user interface, that helps users easily create Blown-up sessions.

Finally I have collected those features of Blown-up that should need correction or fur-

---

ther development, in order to make this middleware more useful. I have presented those properties of BUMP that should only be implemented: the optional pins and the different transport types. I have mentioned the properties that should be designed into BUMP: authentication, authorization, encryption, standardization. I have also described those features that should be modified as differentiation of applications, changing focus, and session handling. I have also describe the possible Blown-up future work, that should enable Blown-up to support *pervasive* applications.

# Appendix A

## API Functions and Structures

### A.1 Description of API Functions

API functions	Description
function:	<code>bump_RegisterApplicationStart</code>
parameters:	<code>string AppName, string AppDescr</code>
return value:	<code>success / failure</code>
function:	<code>bump_RegisterPIN</code>
parameters:	<code>struct PIN</code>
return value:	<code>success / failure</code>
function:	<code>bump_RegisterApplicationEnd</code>
parameters:	<code>-</code>
return value:	<code>success / failure</code>
function:	<code>bump_SendData</code>
parameters:	<code>string MyPINName, void* data, int length</code>
return value:	<code>-1, if pin is not connected 0, if pin is disabled &gt;0, number of sent bytes</code>

function name:	<code>bump_ReadData</code>
parameters:	<code>string MyPINName, void* buffer, int maxbufsize, int Wait</code>
return value:	<code>-1, if pin is not connected &gt;0, number of received bytes</code>
function:	<code>bump_RevokeService</code>
parameters:	<code>-</code>
return value:	<code>void</code>
function:	<code>bump_RegisterControlEntity</code>
parameters:	<code>string controlappname</code>
return value:	<code>success/failure</code>
function:	<code>bump_GetApplicationsReset</code>
parameters:	<code>-</code>
return value:	<code>void</code>
function:	<code>bump_GetApplications</code>
parameters:	<code>struct AppItem*</code>
return value:	<code>0, if there is no more item 1, else</code>
function:	<code>bump_GetPINsReset</code>
parameters:	<code>struct AppItem</code>
return value:	<code>void</code>
function:	<code>bump_GetPINs</code>
parameters:	<code>struct UserPIN*</code>
return value:	<code>0, if there is no more item 1, else</code>
function:	<code>bump_GetInfo</code>
parameters:	<code>struct AppItem, string info, int infolength</code>
return value:	<code>int length</code>
function:	<code>bump_CreateSessionStart</code>
parameters:	<code>int numofpairs</code>
return value:	<code>-1 Failure / bindID</code>

function:	bump_BindPINs
parameters:	struct PINPair, int bindID
return value:	success / failure
function:	bump_CreateSessionEnd
parameters:	int bindID
return value:	success / failure
function:	bump_ChangeRemoteFocus
parameters:	address Addr, int AppID, string MyPINName
return value:	success / failure
function:	bump_ChangeLocalFocus
parameters:	int AppID, string MyPINName
return value:	success / failure
function:	bump_UnRegisterControlEntity
parameters:	–
return value:	void

## A.2 Description of API Structures

API structures	Description
structure:	PIN
elements:	enum TP-Type, int Pin-Type, bool I/O, bool O/M, int Priority, int Capacity, string MyPINName
structure:	UserPIN
elements:	enum TP-Type, int Pin-Type, bool I/O, bool O/M, int Capacity, string MyPINName
structure:	AppItem
elements:	address ServiceAddr, int ServiceAppID
structure:	PINPair
elements:	addr Link1Addr, addr Link2Addr, int Link1AppID addr Link2AppID, string Link1MyPINName, string Link2MyPINName

# Bibliography

- [TDK2002] G. Biczók, K. Fodor, B. Kovács, Á. Szabó: *Blown-up rendszer tervezése és megvalósítása*, Advisors: M. Rónai, Z. Turányi, A. Valkó, Students' Scientific Conference, November 2002.
- [Fodor2003] K. Fodor: *Implementation of a Protocol Stack for Personal Area Networks*, Master's Thesis, May 2003
- [Weiser91] Mark Weiser: *"The Computer for the 21st Century"*, Scientific American, September 1991.
- [Weiser93] Mark Weiser: *"Some Computer Science Issues in Ubiquitous Computing"*, Communications of the ACM, July 1993.
- [Gilder93] George Gilder: *"Dark Fibre, Dark Network"*, Forbes ASAP, December 1993.
- [Tatai97] P. Tatai: *"Open Vocabulary Speech Recognition - Brief State Report on a Research Project"*, Proceedings of the Polish-Czech-Hungarian Workshop on Circuits Theory, Signal Processing and Applications, September 3-7, 1997, Budapest, pp. 52-57.
- [OGN92] G. Olaszy, G. Gordos and G. Németh: *"The MULTIVOX multilingual text-to-speech converter"*, in: G. Bailly, C. Benoit and T. Sawallis (eds.): *Talking machines: Theories, Models and Applications*, Elsevier, 1992, pp. 385-411.
- [RC93] T. Roska and L. O. Chua: *"The CNN Universal Machine: An analogic array computer"*, IEEE Transactions on Circuits and Systems-II, Vol. 40, pp. 163-173, March 1993.
- [WLAN99] *IEEE Std 802.11, 1999 Edition*, <http://standards.ieee.org/catalog/olis/lanman.html>
- [HLAN2] *HiperLAN2 overview*, <http://www.hiperlan2.com/WhyHiperlan2.asp>
- [JH98] Jaap Hartsen: *"BLUETOOTH - The universal radio interface for ad hoc, wireless connectivity"*, Ericsson Review No. 3, 1998.
- [BBSpec] *Bluetooth Baseband Specification*, <http://www.bluetooth.com>
- [Karn90] Phil Karn: *"MACA - A New Channel Access Method for Packet Radio"*, appeared in the proceedings of the 9th ARRL Computer Networking Conference, London, Ontario, Canada, 1990.
- [DPR00] S. Das, C. Perkins, E. Royer: *"Performance Comparison of Two On-demand Ad hoc Routing Algorithms"*, Proceedings of the IEEE Conference on Computer Communication, March 2000.
- [PB94] C. Perkins, P. Bhagwat: *"Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers"*, SIGCOMM'94.



- [PC97] Vincent D. Park and M. Scott Corson: *"A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks"*, Proceedings of IEEE INFOCOM '97, Kobe, Japan (April 1997.)
- [BMJHJ98] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, J. Jetcheva: *"A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols"*, MobiCom '98.
- [McCrory00] Anne McCrory: *"Ubiquitous? Pervasive? Sorry, they don't compute"*, Computer World, March 2000.
- [Satya01] M. Satyanarayanan: *"Pervasive Computing: Vision and Challenges"*, IEEE Personal Communications, August 2001.
- [BPT96] P. Bhagwat, C. Perkins, S. Tripathi: *"Network Layer Mobility: an Architecture and Survey"*, Personal Communications Magazine, Vol. 3, No. 3, June 1996.
- [BSAK95] H. Balakrishnan, S. Seshan, E. Amir, R. Katz: *"Improving TCP/IP Performance Over Wireless Networks"*, MobiCom '95.
- [Oxygen02] *"MIT Project Oxygen"*, Online Documentation,  
<http://oxygen.lcs.mit.edu/publications/Oxygen.pdf>
- [EHAB99] M. Esler, J. Hightower, T. Anderson, G. Borriello *"Next Century Challenges: Data-Centric Networking for Invisible Computing - The Portolano Project at the University of Washington"*, MOBICOM'99.
- [GRIMM et al.01] R. Grimm, J. Davis, E. Lemar, A. MacBeth, S. Swanson, S. Gribble, T. Anderson, B. Bershad, G. Borriello, D. Wetherall. *"Programming for pervasive computing environments"* Technical report UW-CSE-01-06-01, University of Washington, Department of Computer Science and Engineering, June 2001.